



**ТЕХНОЛОГІЇ
ОБ'ЄКТНО-
ОРІЄНТОВАНОГО
ПРОГРАМУВАННЯ**

**ЧАСТИНА I
КОМП'ЮТЕРНИЙ ПРАКТИКУМ**

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

ТЕХНОЛОГІЇ ОБ'ЄКТНО- ОРІЄНТОВАНОГО ПРОГРАМУВАННЯ

ЧАСТИНА І КОМП'ЮТЕРНИЙ ПРАКТИКУМ

*Рекомендовано Методичною радою НТУУ «КПІ ім. Ігоря Сікорського»
як навчальний посібник для студентів,
які навчаються за спеціальністю 151 «Автоматизація та комп'ютерно-
інтегровані технології»*

Київ
КПІ ім. Ігоря Сікорського
2018

ТЕХНОЛОГІЇ ОБ'ЄКТНО-ОРІЄНТОВАНОГО ПРОГРАМУВАННЯ: ЧАСТИНА І. КОМП'ЮТЕРНИЙ ПРАКТИКУМ [Електронний ресурс]: навч. посіб. для студ. спеціальності 151 – «Автоматизація та комп'ютерно-інтегровані технології» / КПІ ім. Ігоря Сікорського; уклад.: В. І. Бендюг, Б. М. Комариста. – Електронні текстові дані (1 файл: 2,84 Мбайт). – Київ: КПІ ім. Ігоря Сікорського, 2018. – 225 с.

*Гриф надано Методичною радою КПІ ім. Ігоря Сікорського (протокол № 10 від 21.06.2018 р.)
за поданням Вченої ради інституту/факультету*

Електронне мережне навчальне видання

ТЕХНОЛОГІЇ ОБ'ЄКТНО-ОРІЄНТОВАНОГО ПРОГРАМУВАННЯ

ЧАСТИНА І

КОМП'ЮТЕРНИЙ ПРАКТИКУМ

Укладачі: Бендюг Владислав Іванович, канд. техн. наук, доцент
Комариста Богдана Миколаївна, канд. техн. наук

Відповідальний редактор А. М. Шахновський, канд. техн. наук, доцент

Рецензенти: Косогін О.В., канд. техн. наук, доцент

©КПІ ім. Ігоря Сікорського, 2018

Зміст

Вступ	9
Комп'ютерний практикум №1	11
Створення консольного додатку	11
Теоретичні відомості	11
Тема 1.1. Створення простої програми на C++	11
1.1.1 Функція <code>main()</code>	11
1.1.2 Файли вихідного коду	12
1.1.3 Компіляція та запуск	12
Тема 1.2. Операції введення-виведення та коментарі	13
1.2.1 Бібліотека <code>iostream</code>	13
1.2.2 Директива <code>#include</code>	13
1.2.3 Оператор виведення <code><<</code>	14
1.2.4 Маніпулятор <code>endl</code>	14
1.2.5 Оператор області видимості <code>::</code>	15
1.2.6 Ініціалізація змінних	15
1.2.7 Оператор введення <code>>></code>	15
Тема 1.3. Форматоване введення-виведення в C++	16
1.3.1 Управління форматом введення-виведення	16
1.3.2 Форматування введення-виведення за допомогою функцій-членів	16
1.3.3 Форматування введення-виведення за допомогою прапорів	18
1.3.4 Форматування введення-виведення за допомогою маніпуляторів	21
Приклад програмної реалізації	24
Завдання до комп'ютерного практикуму №1	25
Контрольні питання	28
Комп'ютерний практикум №2	29
Оператор <code>if</code> та логічні оператори	29
Теоретичні відомості	29
Тема 2.1. Вирази та оператори	29
2.1.1 Унарні та парні оператори	29
2.1.2 Арифметичні оператори	30
2.1.3 Особливості використання операції <code>%</code> та <code>/</code>	30
2.1.4 Пріоритет арифметичних операторів	32

2.1.5 Логічні оператори та оператори відношення	32
2.1.6 Складені оператори присвоєння.....	33
2.1.7 Оператори інкременту та декременту.....	33
2.1.8 Оголошення using.....	34
2.1.9 Використання математичних функцій	34
Тема 2.2. Засоби керування ходом виконання програми	35
2.2.1 Оператори керування потоком	35
2.2.2 Оператори умовного виконання.....	36
Приклад програмної реалізації.....	41
Завдання до комп'ютерного практикуму №2	43
Контрольні питання	49
Комп'ютерний практикум №3	50
Засоби керування процесом виконання програми.....	50
Теоретичні відомості	51
Тема 3.1. Ітераційні оператори.....	51
3.1.1 Оператор while.....	51
3.1.2 Оператор do while	52
3.1.3 Оператор for	54
Приклад програмної реалізації.....	57
Завдання до комп'ютерного практикуму №3	61
Контрольні питання	68
Комп'ютерний практикум №4	69
Змінні та базові типи	69
Теоретичні відомості	70
Тема 4.1. Складені типи	70
4.1.1 Посилання	70
4.1.2 Показчик.....	71
4.1.3 Специфікатор const.....	73
Тема 4.2. Робота з типами.....	74
4.2.1 Псевдоніми типів	74
4.2.2 Специфікатором типу auto	76
4.2.3 Специфікатором типу decltype	76
Тема 4.3. Визначення власних структур даних	77

4.3.1 Ключове слово struct	77
4.3.2 Файли заголовку header	79
4.3.3 Захист заголовку: директиви #define, #ifdef та #ifndef	79
Приклад програмної реалізації	81
Завдання до комп'ютерного практикуму №4	86
Контрольні питання	96
Комп'ютерний практикум №5	97
Типи string та масиви	97
Теоретичні відомості	98
Тема 5.1. Бібліотечний тип string	98
5.1.1 Ініціалізація об'єктів класу string	98
5.1.2 Операції класу string	98
5.1.3 Функції для роботи з розміром рядка string	99
5.1.4 Доступ до елементів рядка string	100
5.1.5 Зміна елементів рядка string	101
5.1.6 Зчитування невизначеної кількості рядків	101
5.1.7 Функція getline()	102
5.1.8 Тип size_type класу string	103
5.1.9 Функції для роботи з окремими символами рядка	103
Тема 5.2. Доступ до окремих елементів	104
5.2.1 Оператор індексування	104
5.2.2 Використання ітераторів	104
5.2.3 Перелік функцій для роботи з ітераторами	104
5.2.4 Операції з ітераторами	107
Тема 5.3. Масиви	108
5.3.1 Оголошення та ініціалізація масиву	108
5.3.2 Звернення до елементів масиву	109
5.3.3 Масиви масивів	110
5.3.4 Динамічні масиви	111
Приклад програмної реалізації	112
Завдання до комп'ютерного практикуму №5	118
Контрольні питання	122
Комп'ютерний практикум №6	123

Тип vector	123
Теоретичні відомості	124
Тема 6.1. Бібліотечний тип vector	124
6.1.1 Визначення і ініціалізації векторів	124
6.1.2 Функція push_back()	126
6.1.3 Операції з векторами	126
6.1.4 Функції empty() та size()	127
6.1.5 Використання ітераторів з векторами	127
Тема 6.2. Генератор випадкових чисел	129
6.2.1 Генерування цілих випадкових чисел в заданому діапазоні	129
6.2.2 Генерування випадкових чисел з плаваючою комою в заданому діапазоні	130
6.2.3 Використання функції time() для генерування випадкових чисел	131
Приклад програмної реалізації	133
Завдання до комп'ютерного практикуму №6	143
Контрольні питання	149
Комп'ютерний практикум №7	150
Оператори	150
Теоретичні відомості	151
Тема 7.1. Оператори зміни порядку виконання програми	151
7.1.1 Умовний оператор (? :)	151
7.1.2 Оператор switch	152
7.1.3 Серійний оператор for	154
7.1.4 Оператори переходу	157
Тема 7.2. Виключення	165
7.2.1 Оператор throw та блок try	165
7.2.2 Стандартні класи виключень	168
Приклад програмної реалізації	171
Завдання до комп'ютерного практикуму №7	177
Контрольні питання	185
Комп'ютерний практикум №8	186
Функції	186
Теоретичні відомості	187
Тема 8.1. Функції	187

8.1.1 Физначення та виклик функції	187
8.1.2 Аргументи та перелік параметрів функції.....	188
8.1.3 Область видимості імен	189
8.1.4 Оголошення функції.....	190
8.1.5 Передача параметрів за посиланням та за значенням	191
Приклад програмної реалізації.....	193
Завдання до комп'ютерного практикуму №8	198
Контрольні питання	200
Комп'ютерний практикум №9	201
Класи.....	201
Теоретичні відомості	202
Тема 9.1. Класи.....	202
9.1.1 Абстракція даних та інкапсуляція	202
9.1.2 Змінні-члени та функції-члени	203
9.1.3 Оператор доступу до члену та оператор області видимості	204
9.1.4 Конструктори	205
9.1.5 Дружні відносини.....	207
Приклад програмної реалізації.....	209
Завдання до комп'ютерного практикуму №9	218
Контрольні питання	220
Рекомендована література	221
Додаток А.....	223
Формули для довідок.....	223

Вступ

Згідно робочого навчального плану кредитний модуль «Прикладне програмне забезпечення - 2. Технології об'єктно-орієнтованого програмування» дисципліни «Прикладне програмне забезпечення» викладається студентам четвертого року підготовки ОКР «бакалавр» спеціальності 151 - Автоматизація та комп'ютерно-інтегровані технології у сьомому навчальному семестрі. Матеріал кредитного модуля базується на дисциплінах «Комп'ютерні технології та програмування - 1. Основи алгоритмізації», «Комп'ютерні технології та програмування - 2. Програмування типових задач», «Комп'ютерна техніка та організація обчислювальних робіт», «Комп'ютерні технології та програмування - 3. Розробка інтерфейсу користувача», «Інформаційні системи та комплекси». Знання уміння та навички, одержані під час вивчення даного модуля, у подальшому використовуються в усіх дисциплінах, які потребують програмної реалізації розрахунків на комп'ютері, і в першу чергу – в курсах «Ідентифікація та моделювання об'єктів автоматизації», «Основи проектування систем автоматизації і систем керування експериментом», «Методи штучного інтелекту та їх застосування в хімічній технології», «Прикладне програмне забезпечення - 3. Проектування програмних доданків», в курсових і дипломних роботах і проектах.

Метою навчальної дисципліни є формування у студентів здібностей:





КСР-24 - вміння застосовувати комп'ютерну техніку для вирішення технічних задач;

КСР-25 - вміння використовувати комп'ютерно-інтегровані технологічні та інформаційні системи;

КСР-26 – вміння розробляти, тестувати та застосовувати програмне забезпечення для вирішення прикладних задач.

Згідно з вимогами освітньо-професійної програми студенти після засвоєння навчальної дисципліни мають продемонструвати такі результати навчання:

знання:

-  стандартів сучасних об'єктно-орієнтованих мов програмування;
-  інтерфейсу середовища програмування Microsoft Visual Studio;
-  сучасних методів конструювання програм в тому числі засобами об'єктно-орієнтованої мови програмування C++;
-  ефективних методів збереження і обробки інформації;

уміння:

- 🖥️ самотійно розробляти ефективні алгоритми і програми на сучасній алгоритмічній мові для вирішення поставленої задачі;
- 🖥️ налагоджувати програми на ПК до надійної працездатності в ОС Windows;

досвід:

- 🖥️ виконувати налаштування середовища програмування для ефективної роботи;
- 🖥️ оформляти розроблені програми згідно вимогам ЄСПД.



Комп'ютерний практикум №1

Створення консольного додатку

Мета: ознайомитись з навичками роботи в командному режимі та принципами створення *консольного додатку* C++. Засвоїти методи *компіляції, запуску та відлагоджування* програми. Вивчити оператори *потоків введення-виведення* в консольному додатку.

Завдання: створити консольний додаток для відпрацювання основних навичок створення та *компіляції* додатку, а також реалізувати поставлену задачу згідно отриманого варіанту завдання.

Загальні вимоги.

- 1) Створити консольний додаток з ім'ям виду *PrizvischeKP1*.
- 2) Програмний код модуля має бути чітко структурований.
- 3) Імена об'єктів мають нести сенсові навантаження.
- 4) Програмний код має супроводжуватись коментарями в тексті програми.

Вимоги до виконання.

- 1) Створити новий проект Visual C++ типу Win32 Console Application.
- 2) Розробити алгоритм розрахунку залежності згідно свого варіанту.
- 3) Підключити бібліотеку *потоків введення-виведення iostream*.
- 4) Організувати введення вхідних даних за допомогою об'єкту *cin* стандартної бібліотеки *std*.
- 5) Організувати виведення запитів користувачу та результатів розрахунку за допомогою об'єкту *cout* стандартної бібліотеки *std*.
- 6) Вставити в тексті програми *коментарі* для пояснення її роботи.
- 7) Встановити в тексті програми *точку зупинки* і у *покроковому режимі* переглянути поточні значення змінних.

Теоретичні відомості

Тема 1.1. Створення простої програми на C++

1.1.1 Функція `main()`

Кожна програма C++ містить одну чи декілька *функцій* (function), при цьому одна з них обов'язково має ім'я `main()`. Запускаючи програму C++ операційна система викликає саме функцію `main()`.

Визначення функції містить чотири елементи: *тип поверненого значення* (return type), *ім'я функції* (function name), *список параметрів* (parameter list), який може бути пустий, і *тіло функції* (function body).

```
тип поверненого значення ім'я функції(список параметрів)
{
    Тіло функції;
}
```

Функція `main()` повинна мати тип поверненого значення `int`, який є типом цілих чисел. Тип `int` – це *вбудований тип* (built-in type) даних, такі типи визначені у самій мові.

Заклучна частина визначення функції, її тіло, являє собою блок операторів (block of statements), який починається відкриваючою *фігурною дужкою* (curly brace) та завершується закриваючою фігурною дужкою.

```
int main()
{
    return 0;
}
```

Оператор `return` завершує код функції. Він може передати значення назад стороні, яка викликала функцію. Значення, яке отримує оператор `return`, за типом має сумісним з типом поверненого значення функції.

Більшість операторів мови C++ завершуються крапкою з комою. Її відсутність призводить до виведенні компілятором повідомлень про помилки.

В більшості операційних систем повернене функцією `main()` значення використовується як індикатор стану. Повернене значення `0` свідчить про успіх. Будь-яке інше значення, як правило, означає відмову, а саме значення вказує на її причину.

1.1.2 Файли вихідного коду

Файли програм зазвичай називають *файлами вихідного коду* (source file). Ім'я файлу завершується суфіксом (розширенням). Різні компілятори використовують суфікси файлів вихідного коду; до найбільш поширених відносять `.cc`, `.cxx`, `.cpp`, `.sr` та `.C`.

1.1.3 Компіляція та запуск

Написану програму необхідно відкомпілювати. Спосіб компіляції програми залежить від операційної системи та компілятора. Більшість PC-орієнтованих компіляторів мають *інтегроване середовище розробки* (Integrated

Development Environment - IDE), яке об'єднує компілятор з відповідними засобами редагування та налагодження коду.

Спосіб компіляції програми залежить від компілятора, який використовується, а також від обраних опцій. Крім того, більшість компіляторів, наприклад Microsoft Visual C++, надають два різних способи компіляції програм: за допомогою компілятора командного рядка і інтегрованого середовища розробки IDE.

В наслідок роботи C++ - компілятора отримуємо виконуваний об'єктний код. Для Windows-середовища виконуваний файл матиме теж ім'я, що і вихідний, але інше розширення файлу, а саме *.exe.

Тема 1.2. Операції введення-виведення та коментарі

1.2.1 Бібліотека iostream

В самій мові C++ немає операторів для *введення та виведення* (Input/Output - IO). Їх функції забезпечує *стандартна бібліотека* (standard library). Основу *бібліотеки iostream* складають два типи, *istream* та *ostream*, які представляють потоки введення та виведення відповідно. *Потік* (stream) – це послідовність символів, яка записується чи зчитується з пристрою введення-виведення деяким способом. Термін «потік» передбачає, що символи надходять та передаються послідовно протягом певного часу.

В бібліотеці визначені чотири об'єкти введення-виведення. Для здійснення введення використовують об'єкт *cin* (вимовляється «сі-ін») типу *istream*. Цей об'єкт називають також *стандартним введенням* (standard input). Для виведення використовується об'єкт *cout* (вимовляється «сі-аут») типу *ostream*. Його часто згадують як *стандартне виведення* (standard output).

В бібліотеці визначені ще два об'єкти типу *ostream* – це *cerr* та *clog* (вимовляється як «сі-ерр» та «сі-лог» відповідно). Об'єкт *cerr* називають також *стандартною помилкою* (standard error) та, як правило, використовують в програмах для створення попереджень і повідомлень про помилки, а об'єкт *clog* – для створення інформаційних повідомлень.

1.2.2 Директива #include

Для включення компілятором в програму бібліотеки *iostream* використовується *директива препроцесору* (preprocessor directive)

```
#include <iostream>
```


Ім'я в кутових дужках – це *заголовок* (header). Кожна програма, яка використовує засоби, що зберігаються в бібліотеці, має підключити відповідний заголовок. Директива `#include` має бути написана в одному рядку та знаходитись поза тілом функції. Зазвичай всі директиви `#include` програми розташовані на початку файлу вихідного коду.

1.2.3 Оператор виведення <<

Перший оператор в тілі функції `main()` виконує *вираз* (expression). В мові C++ вираз складається з одного чи кількох *операндів* (operand) і, як правило, *оператора* (operator). Щоб відобразити підказку на стандартному пристрої виведення, використовується *оператор виведення* (output operator), або *оператор <<*.

```
std::cout << "Input t1" << std::endl;
```

В даному рядку об'єднані два оператори виведення. Перший оператор `std::cout << "Input t1"` виводить повідомлення для користувача. Це повідомлення є *рядковим літералом* (string literal) – послідовність символів, які заключені у подвійні лапки. Оператор складається з двох операндів, лівий повинен бути об'єктом класу `ostream`, а правий операнд – це значення для відображення. Оператор `<<` заносить передане значення в об'єкт `cout` класу `ostream`. Таким чином, результатом буде об'єкт класу `ostream`, в який записане передане значення. Текст у подвійних лапках виводиться на стандартний пристрій виведення.

1.2.4 Маніпулятор endl

Другий оператор в наведеному вище прикладі `std::cout << std::endl` виводить `endl` (читається як «енд-ел») – спеціальне значення, що зветься *маніпулятором* (manipulator). При його запису у потік виведення здійснюється перехід на новий рядок та скидання *буферу* (buffer), який пов'язаний з даним пристроєм. Скидання буферу гарантує, що все виведення, яке програма сформувала на даний момент, буде одразу записане в потік виведення, а не буде очікувати запису, знаходячись у пам'яті.

Оскільки перша частина оператора повертає об'єкт класу `ostream`, то два оператори

```
std::cout << "Input v1 and v2";  
std::cout << std::endl;
```

можна об'єднати, що і було зроблено.

1.2.5 Оператор області видимості ::

В наведених операторах використана форма запису `std::cout` та `std::endl` замість `cout` та `endl`. Префікс `std::` означає, що імена `cout` та `endl` визначені у просторі імен (namespace) на ім'я `std`. Простори імен дозволяють уникнути імовірних конфліктів, причинами яких може стати співпадіння імен, визначених в різних бібліотеках. Всі імена, які визначені в стандартній бібліотеці, знаходяться у просторі імен `std`. В запису `std::cout` застосовується *оператор області видимості ::* (scope operator), який вказує що тут використовується ім'я `cout`, яке визначене в просторі імен `std`.

1.2.6 Ініціалізація змінних

Після запиту на введення даних, необхідно організувати читання введених користувачем даних. Але перед цим потрібно визначити *змінні* (variable), в які будуть запам'ятовуватись введені значення.

```
int v1 = 0, v2=0;
```

Ці змінні визначені як такі, що відносяться до типу `int`. Тип `int` є вбудованим типом даних для цілочисельних значень. Окрім того, ми також *ініціалізуємо* (initialize) змінні, тобто присвоюємо вказане значення на момент створення.

1.2.7 Оператор введення >>

Оператор введення >> (input operator) поводить себе аналогічно оператору виведення. Його лівим операндом (відносно оператору >>) є об'єкт класу `istream`, а правим операндом – об'єкт, який запам'ятовує отримані дані. Подібно оператору виведення, оператор введення повертає як результат свій лівий операнд (об'єкт класу `istream`), тому можна об'єднувати кілька операторів введення

```
std::cin >> v1 >> v2;
```

що рівноцінно

```
std::cin >> v1;  
std::cin >> v2;
```


Тема 1.3. Форматоване введення-виведення в C++

1.3.1 Управління форматом введення-виведення

Можливість управляти форматом введення-виведення в C++, забезпечують функції-члени, прапори і маніпулятори. Прапори, функції та маніпулятори виконують одну й ту ж саму задачу - задають певний формат введення/виведення інформації в потоках. Введення/виведення на екран / з екрану в C++ здійснюється за допомогою операторів `cin` і `cout` відповідно, а значить маніпулятори форматування використовуються спільно з цими операторами введення/виведення. Різниця між функціями, прапорами і маніпуляторами форматування полягає в способі їх застосування. Тепер розглянемо способи застосування об'єктів форматування.

1.3.2 Форматування введення-виведення за допомогою функцій-членів

В наступному прикладі наведені основні функції форматування введення/виведення.

```
// Основні функції-члени форматування:  
cout.fill('/* symbol */'); // Встановлює символ заповнювач,  
// де symbol - символ заповнювач, символ передається в одинарних лапках  
cout.width(/* width_field */); // Задає ширину поля,  
// де width_field - кількість позицій (одна позиція вміщує один символ)  
cout.precision(/* number */); // Задає максимальну кількість цифр для  
// відображенн, де number - кількість цифр
```

Доступ до функцій здійснюється через оператор *точка*, а в круглих дужках передається аргумент. Аргумент функції *fill()* може передаватися у вигляді символу, обрамленого одинарними лапками, або у вигляді числа (код символу).

Функція *width()* задає ширину поля, тобто наступне виведення буде мати вказану у параметрі функції *width()* загальну довжину разом з усіма виведеними символами. Якщо кількість символів менша ніж вказано у функції *width()*, то поле доповниться потрібною кількістю пробілів перед виведеним значенням. При використанні функції *width()* у парі з функцією *fill()* можна задати інші символи замість пробілів, якими буде доповнюватись поле виведення.

```
double j = 123.45;  
std::cout.fill('*'); // Встановлює символ заповнювач  
std::cout.width(10); // Задає ширину поля  
std::cout << j;
```

Результатом роботи даного коду буде виведене наступне значення

****123.45

Тобто наше дійсне число 123.45, яке складається з шести символів разом з крапкою, доповнене чотирма вказаними у функції `fill()` символами `*` до загальної кількості символів – десять.

Функція `precision()` задає кількість цифр, які будуть відображатися у числа з плаваючою точкою в наступному операторі виведення. При цьому рахуються всі символи як до десяткової точки, так і після неї. Для прикладу розглянемо наступний код:

```
double i = 12.345678;
std::cout.precision(3);
std::cout << i << std::endl;
std::cout.precision(5);
std::cout << i << std::endl;
std::cout.precision(10);
std::cout << i << std::endl;
```

Результатом буде наступне виведення:

```
12.3
12.346
12.345678
```

В першому випадку `precision(3)` встановлена кількість символів 3 і відповідно виводиться значення 12.3, тобто значення округлене до трьох символів, інші відкидаються. В другому випадку `precision(5)` встановлена кількість символів 5 і відповідно виводиться значення 12.346, тобто значення округлюється до п'яти символів, інші відкидаються. В третьому випадку `precision(10)` встановлена кількість символів 10 більша ніж кількість символів у значенні з плаваючою точкою і відповідно виводиться значення 12.345678, тобто значення з меншою кількістю символів ніж задано у функції `precision()`.

Результат роботи функції `precision()` залежить від заданого формату виведення числових значень. Формат виведення числових значень можна задати за допомогою функції `setf()`.

Недоліком функцій форматування введення/виведення є те, що кожна функція зазвичай *спрацьовує лише один єдиний раз на наступному операторі виведення*, після чого формат виведення повертається до форматування за замовчуванням.

Одних функцій мало для форматування потоків введення/виведення, тому в C++ передбачено ще один спосіб форматування - прапори.

Функція `setf()` встановлює прапори форматування чи прапори форматування та маску формату:

```
setf(прапор)
setf(прапор, маска)
```

При цьому комбінація прапорів та масок може бути наступною (табл. 1.3.1).

Таблиця 1.3.1 –Прапори та маски функції `setf()`

Маска	Прапори
<code>adjustfield</code>	<code>left</code> , <code>right</code> або <code>internal</code>
<code>basefield</code>	<code>dec</code> , <code>oct</code> або <code>hex</code>
<code>floatfield</code>	<code>scientific</code> або <code>fixed</code>

Призначення прапорів наведено в табл. 1.3.2.

Наступний приклад дозволяє доповнювати нулями кількість символів дійсного числа до значення вказаного у функції `precision()` за допомогою встановленого формату у функції `setf()`.

```
double i = 12.345678;
std::cout.setf(std::ios::fixed, std::ios::floatfield);
std::cout.precision(10);
std::cout << i << std::endl;
```

Результатом роботи даного коду буде наступне виведення:

```
12.3456780000
```

Тобто при встановленій масці `floatfield` та формату чисел з плаваючою точкою `fixed` функція `precision()` тепер вказує не загальну кількість символів в числі з плаваючою точкою, а кількість символів після коми. При цьому число вже буде доповнюватись нулями, якщо кількість символів після коми менша ніж вказано у функції `precision()`.

Функція `unsetf()` відмінняє певний режим форматування.

1.3.3 Форматування введення-виведення за допомогою прапорів

Прапори форматування дозволяють включити або виключити один з параметрів введення/виведення. Щоб встановити прапор введення/виведення, необхідно викликати функцію `setf()`, якщо необхідно відключити прапор виведення, то використовується функція `unsetf()`. Далі показані конструкції установки і зняття прапорів виведення.


```
// Установка прапора виведення
cout.setf(ios :: /* name flag */);
// Де name_flag - це ім'я прапора
```

Доступ до функцій оператора виведення виконується через оператор *крапка*. Метод (функція) `setf()` приймає один аргумент - ім'я прапора. Прапори виведення оголошені в класі `ios`, тому, перед тим, як звернутися до прапора, необхідно написати ім'я класу - `ios`, після якого, за допомогою *оператора дозволу області дії* (`::`), викликати потрібний прапор.

```
// Зняття прапора виведення
cout.unsetf(ios :: /* name flag */);
// Де name_flag - це ім'я прапора
```

Якщо при введенні/виведенні необхідно встановити (зняти) кілька прапорів, то можна скористатися *порозрядною логічною операцією АБО* `|`. У цьому випадку конструкція мови C++ буде такою:

```
// Установка декількох прапорів
cout.setf(ios :: /* name_flag 1 */ | ios :: /* name_flag 2 */ | ios ::
/* name_flag_n */);
// Зняття декількох прапорів
cout.unsetf(ios :: /* name_flag1 */ | ios :: /* name_flag2 */ | ios ::
/* name_flag_n */);
```

У таблиці 1.3.2 детально описані основні прапори форматування, а також показані приклади використання прапорів.

Таблиця 1.3.2 –Прапори форматування в C++

Прапор	Призначення	Приклад	Результат
boolalpha	Виведення логічних величин в текстовому вигляді (true, false)	<pre>cout.setf(ios::boolalpha); bool log_false = 0, log_true = 1; cout << log_false << endl << log_true << endl;</pre>	false true
oct	Введення/виведення величин в вісімковій системі числення (спочатку знімаємо прапор <code>dec</code> , потім встановлюємо прапор <code>oct</code>)	<pre>cout.unsetf(ios::dec); cout.setf(ios::oct); int value; cin >> value; cout << value << endl;</pre>	введення: 99 ₁₀ виведення: 143 ₈
dec	Введення/виведення величин в десятковій системі числення	<pre>cout.setf(ios::dec); int value = 148; cout << value << endl;</pre>	148

Прапор	Призначення	Приклад	Результат
	(прапор встановлений за замовчуванням)		
hex	Введення/виведення величин в шістнадцятковій системі числення (спочатку знімаємо прапор dec, потім встановлюємо прапор hex)	<pre>cout.unsetf(ios::dec); cout.setf(ios::hex); int value; cin >> value; cout << value << endl;</pre>	введення:99 ₁₀ виведення:63 ₁₆
showbase	Виводити індикатор основи системи числення	<pre>cout.unsetf(ios::dec); cout.setf(ios::oct ios::showbase); int value; cin >> value; cout << value << endl;</pre>	введення:99 ₁₀ виведення:0143 ₈
uppercase	У шістнадцятковій системі числення використовувати літери верхнього регістру (за замовчуванням встановлено літери нижнього регістра)	<pre>cout.unsetf(ios::dec); cout.setf(ios::hex ios::uppercase); int value; cin >> value; cout << value << endl;</pre>	введення:255 ₁₀ виведення:FF ₁₆
showpos	Виведення знака плюс + для позитивних чисел	<pre>cout.setf(ios::showpos); int value = 15; cout << value << endl;</pre>	+15
scientific	Виведення чисел з плаваючою точкою в експоненційній формі	<pre>cout.setf(ios::scientific); double value = 1024.165; cout << value << endl;</pre>	1.024165e+003
fixed	Виведення чисел з плаваючою точкою у фіксованій формі (за замовчуванням)	<pre>double value = 1024.165; cout << value << endl;</pre>	1024.165
right	Вирівнювання праворуч (за замовчуванням) та додає на початку символи	<pre>cout.setf(ios::right); cout.width(40); cout << "cppstudio.com" << endl;</pre>	__cppstudio.com

Прапор	Призначення	Приклад	Результат
	заповнення. Спочатку необхідно встановити ширину поля (ширина поля повинна бути свідомо більшою ніж, довжина рядка, що виводиться).		
left	Вирівнювання ліворуч та додає в кінець символи заповнення. Спочатку необхідно встановити ширину поля (ширина поля повинна бути свідомо більшою ніж, довжина рядка, що виводиться).	<pre>cout.setf(ios::left); cout.width(40); cout << "cppstudio.com" << endl;</pre>	cppstudio.com__

1.3.4 Форматування введення-виведення за допомогою маніпуляторів

Ще один спосіб форматування - форматування за допомогою маніпуляторів. *Маніпулятор* - об'єкт особливого типу, який управляє потоками введення/виведення, для форматування переданої в потоки інформації. Почасти маніпулятори доповнюють функціонал, для форматування введення/виведення. Але більшість маніпуляторів виконують точно, те ж саме, що і функції з прапорами форматування. Є випадки, коли простіше користуватися прапорами або функціями форматування, а іноді зручніше використовувати маніпулятори форматування. Саме для цього в C++ передбачено кілька засобів форматування введення/виведення. У таблиці 1.3.3 показані основні маніпулятори форматування C++.

Таблиця 1.3.3 –Маніпулятори форматування в C++

Маніпулятор	Призначення	Приклад	Результат
endl	Перехід на новий рядок при виведенні	<pre>cout << "website: " << endl << "cppstudio.com";</pre>	website: cppstudio.com
boolalpha	Виведення логічних величин в текстовому	<pre>bool log_true = 1; cout << boolalpha << log_true << endl;</pre>	true

Маніпулятор	Призначення	Приклад	Результат
	вигляді (true, false)		
noboolalpha	Виведення логічних величин в числовому вигляді (true, false)	<pre>bool log_true = true; cout << noboolalpha << log_true << endl;</pre>	1
oct	Виведення величин в восьмеричній системі числення	<pre>int value = 64; cout << oct << value << endl;</pre>	100₈
dec	Виведення величин в десятковій системі числення (за замовчуванням)	<pre>int value = 64; cout << dec << value << endl;</pre>	64₁₀
hex	Виведення величин в шістнадцятиричній системі числення	<pre>int value = 64; cout << hex << value << endl;</pre>	40₈
showbase	Виводити індикатор основи системи числення	<pre>int value = 64; cout << showbase << hex << value << endl;</pre>	0x40
noshowbase	Не виводити індикатор основи системи числення (за замовчуванням)	<pre>int value = 64; cout << noshowbase << hex << value << endl;</pre>	40
uppercase	У шістнадцятиричній системі числення використовувати літери верхнього регістру (за замовчуванням встановлено літери нижнього регістра)	<pre>int value = 255; cout << uppercase << hex << value << endl;</pre>	FF₁₆
nouppercase	У шістнадцятиричній системі числення використовувати літери нижнього регістра (за замовчуванням)	<pre>int value = 255; cout << nouppercase << hex << value << endl;</pre>	ff₁₆
showpos	Виведення знака плюс + для позитивних чисел	<pre>int value = 255; cout << showpos << value << endl;</pre>	+255
noshowpos	Не виводити знак плюс + для позитивних чисел (за замовчуванням)	<pre>int value = 255; cout << showpos << value << endl;</pre>	255
scientific	Виведення чисел з плаваючою точкою в експоненційній формі	<pre>double value = 1024.165; cout << scientific << value << endl;</pre>	1.024165e+003
fixed	Виведення чисел з плаваючою точкою у фіксованій формі (за замовчуванням)	<pre>double value = 1024.165; cout << fixed << value << endl;</pre>	1024.165

Маніпулятор	Призначення	Приклад	Результат
setw(int number)	Встановити ширину поля, де number - кількість позицій, символів (вирівнювання за замовчуванням праворуч). Маніпулятор з параметром	<pre>cout << setw(40) << "cppstudio.com" << endl;</pre>	__cppstudio. com
right	Вирівнювання за правою межею (за замовчуванням). Спочатку необхідно встановити ширину поля (ширина поля повинна бути свідомо більшою ніж, довжина рядка, що виводиться)	<pre>cout << setw(40) << right << "cppstudio.com" << endl;</pre>	__cppstudio. com
left	Вирівнювання ліворуч. Спочатку необхідно встановити ширину поля (ширина поля повинна бути свідомо більшою ніж, довжина рядка, що виводиться)	<pre>cout << setw(40) << left << "cppstudio.com" << endl;</pre>	cppstudio.co m__
setprecision(int count)	Задає кількість знаків після коми, де cout - кількість знаків після десяткової точки	<pre>cout << fixed << setprecision(3) << (13.5 / 2) << endl;</pre>	6.750
setfill(int symbol)	Встановити символ заповнювач. Якщо ширина поля більше, ніж величина яка виводиться, то вільні місця поля будуть заповнюватися символом symbol - символ заповнювач	<pre>cout << setfill('0') << setw(4) << 15 << ends << endl;</pre>	0015

Форматоване введення/виведення в C++ - це одна з найпростіших тем в програмуванні. Як використовувати ті чи інші засоби форматування показано в таблицях, тому труднощів з даної теми виникнути не повинно.

Приклад програмної реалізації

Приклад створення консольного додатку, використання операторів введення-виведення, додавання коментарів.

```
//Приклад створення консольного додатку та використання операторів
введення-виведення
//

#include "stdafx.h"
#include <iostream>

int main() //Розрахувати об'єм газу за залежністю  $P_1 \cdot V_1 / T_1 = P_2 \cdot V_2 / T_2$ 
{
    /*При заданих значеннях температури в Цельсіях,
    тиску в кПа та об'єму в л газу
    розрахувати об'єм газу при іншій температурі та тиску*/
    float t1, p1, v1, t2, p2, v2;    //Оголошення змінних дійсного типу,
                                   // які використовуються в розрахунках
    const int t0 = 273;    //Оголошення цілочисельної константи для
    //переведення температури з градусів Цельсія в Кельвіни
    std::cout << "Input t1" << std::endl; //Виведення на екран запиту
    //для введення температури
    std::cin >> t1; //Запис введеного з клавіатури значення в змінну t1
    std::cout << "Input P1" << std::endl; //Виведення на екран запиту
    //для введення тиску
    std::cin >> p1; //Запис введеного з клавіатури значення в змінну p1
    std::cout << "Input V1" << std::endl; //Виведення на екран запиту
    //для введення об'єму газу
    std::cin >> v1; //Запис введеного з клавіатури значення в змінну v1
    std::cout << "Input t2" << std::endl; //Виведення на екран запиту
    //для введення температури, при якій необхідно
    //розрахувати об'єм газу
    std::cin >> t2; //Запис введеного з клавіатури значення в змінну t2
    std::cout << "Input P2" << std::endl; //Виведення на екран запиту
    //для введення тиску, при якому необхідно розрахувати об'єм газу
    std::cin >> p2; //Запис введеного з клавіатури значення в змінну p2
    v2 = p1*v1*(t2 + t0) / (p2*(t1 + t0)); //Розрахункова залежність
    //для об'єму газу
    std::cout << "V2" << v2 << std::endl; //Виведення на екран
    //розрахованого об'єму газу
    return 0; //Завершення роботи функції main()
    //і повернення значення 0
}
```


Завдання до комп'ютерного практикуму №1

1. Обчислити об'єм, який займає 5 кг CO_2 при $t = -9^\circ\text{C}$ та $p = 1,64$ атм. (Дивись формулу (1)).
2. Обчислити молекулярну масу хімічної сполуки та масову частку кожного з хімічних елементів, які входять у склад ортоалюмінату кальцію $\text{Ca}(\text{AlO}_3)_2$.
(Використати формулу: $N_i = \frac{n_i \cdot A_i}{\sum n_i A_i} \cdot 100\%$).
3. Обчислити молекулярну масу хімічної сполуки та масову частку кожного з хімічних елементів, які входять у склад метіоніна $\text{CH}_3\text{S}(\text{CH}_2)_2\text{CH}(\text{NH}_2)\text{CO}_2\text{H}$. (Брутто формула $\text{C}_5\text{H}_{11}\text{NO}_2\text{S}$).
4. Знайти брутто формулу хімічної сполуки, якщо відомо, що її молекулярна маса дорівнює $M = 73$, а елементний склад $X_{\text{C}} = 49\%$; $X_{\text{H}} = 10\%$; $X_{\text{N}} = 19\%$; $X_{\text{O}} = 22\%$.
(Використати формулу $n_i = \frac{M \cdot X_i}{A_i}$).
5. Який об'єм ацетилену теоретично можливо спалити у 1 м^3 повітря. Який об'єм CO_2 буде при цьому отримано. (Прийняти, що 1 м^3 повітря має $0,21 \text{ м}^3$ кисню).
6. Знайти масу 200 л хлору при $t = 0^\circ\text{C}$ та $p = 101,3$ кПа. (Дивись формулу (1)).
7. Знайти масу 1,8 л H_2S , якщо відомо, що об'єм газу вимірювали при $t = 17^\circ\text{C}$ та $p = 98,64$ кПа. (Дивись формулу (1)).
8. При $t = 0^\circ\text{C}$ отримали суміш 5 л метану, 10 л водню та 25 л кисню. Розрахувати концентрацію складових частин газової суміші (у об'ємних %, та г/л).
9. Згідно з умовою завдання 8 розрахувати концентрацію у г/л та % за масою.
10. Згідно з умовою завдання 8 розрахувати концентрацію у % за масою та молях на літр.
11. Згідно з умовою завдання 8 розрахувати концентрацію у об'ємних % та молях на літр.
12. Згідно з умовою завдання 8 розрахувати парціальний тиск компонентів, якщо тиск дорівнює 101,3 кПа.
13. Обчислити об'єм, який займає 7 г CO_2 при $t = 7^\circ\text{C}$ та $p = 104$ кПа. (Дивись формулу (1)).

14. Об'єм газу при $t_1 = 23^\circ\text{C}$ та $p_1 = 103,3$ кПа дорівнює 250 л. Знайти об'єм, який займає газ при $t_2 = 0^\circ\text{C}$ та $p_2 = 101,3$ кПа.

(Використати формулу $\frac{P_1 \cdot V_1}{T_1} = \frac{P_2 \cdot V_2}{T_2}$).

15. Розрахувати, скільки молекул вміщає 1 мл газу при $t = 23^\circ\text{C}$ та $p = 2,53$ кПа. (Число Авогадро $A = 6,02 \cdot 10^{23}$).

16. Розрахувати, скільки молекул вміщає 1 мл газу при нормальних умовах ($t = 0^\circ\text{C}$ та $p = 101,3$ кПа). (Число Авогадро $A = 6,02 \cdot 10^{23}$).

17. Маса 1 л газу при нормальних умовах ($t_2 = 0^\circ\text{C}$ та $p_2 = 101,3$ кПа) дорівнює 1,25 г. Розрахувати мольну масу газу, та масу його молекули. (Число Авогадро $A = 6,02 \cdot 10^{23}$, дивись формулу (1)).

18. Концентрація гідроксильних іонів дорівнює $[\text{OH}^-] = 2,5 \cdot 10^{-12}$ моль/л. Розрахувати рН розчину. (Пам'ятати, що $[\text{OH}^-][\text{H}^+] = K_w = 10^{-14}$).

19. Концентрація гідроксильних іонів дорівнює $[\text{OH}^-] = 9 \cdot 10^{-9}$ моль/л. Розрахувати рН розчину. (Пам'ятати, що $[\text{OH}^-][\text{H}^+] = K_w = 10^{-14}$).

20. Знайти кількість молекул, яку вміщує 1 г H_2SO_4 та 1 г HNO_3 . У якій сполуці (у скільки разів) молекул більше? (Число Авогадро $A = 6,02 \cdot 10^{23}$).

21. Скільки років потрібно, щоб перерахувати кількість молекул, яку вміщає 1 г води, якщо рахувати 1 молекулу в 1 секунду (1 рік = 365 днів).

22. Скільки років потрібно, щоб із заповненої гелієм ампули (умови нормальні, $t = 0^\circ\text{C}$; $p = 101,3$ кПа) ємкістю 1 мм^3 повністю евакуувати весь газ із швидкістю 10^6 атомів у секунду. (Число Авогадро $A = 6,02 \cdot 10^{23}$).

23. Розрахувати рівнодіючу двох сил $F_1 = 24 \text{ Н}$ і $F_2 = 38 \text{ Н}$, які діють на одну точку під кутом 48° одна до одної. (Дивись формулу 2.2)

24. Крапля, яка має заряд $q = 1 \cdot 10^{-10}$ кл, розділилася на дві частини, які на відстані $R = 5 \cdot 10^{-10}$ м взаємодіють із силою $F = 1 \text{ Н}$. Знайти заряд кожної частини. ($q_1 + q_2 = q$; $F = 4\pi\epsilon_0 R$; $E = 8,8541878 \cdot 10^{-2} \text{ кл}/(\text{В} \cdot \text{м})$).

25. Ввести два числа А та В. Знайти число С, яке дорівнює В відсотків від числа А.

26. Обчислити площу трикутника та синуси всіх його кутів, якщо відомі сторони трикутника. (Дивись формулу 2.4 та 2.5).

27. Знайти висоти трикутника за його сторонами. (Дивись формули 2.3 та 2.4).
28. За двома сторонами та кутом між ними знайти третю сторону та площу трикутника. (Дивись формули 2.4 та 2.5).
29. Сторони трикутника дорівнюють 4 см, 5 см, 6 см. Знайти площу трикутника, сторони якого є медіанами заданого. (Дивись формули 2.4 та 2.5).
30. Основа рівнобедреного трикутника дорівнює A , а кут між рівними його сторонами – b . Знайти площу трикутника та тригонометричні функції всіх його кутів. (Дивись формули 2.3 та 2.5).



Контрольні питання

- 1) Що таке функція `main()`, з чого вона складається?
- 2) Який тип даних має повертати функція `main()`?
- 3) Що таке вбудований тип даних?
- 4) Що таке тіло функції, з чого воно складається?
- 5) Що таке оператор `return`, для чого він використовується?
- 6) Що таке потік введення чи виведення?
- 7) Що таке стандартне введення та стандартне виведення?
- 8) Що таке об'єкт `cerr` та `clog` і для чого вони використовуються?
- 9) Що таке директива `#include` та для чого вона використовується?
- 10) Що таке оператор виведення та для чого він використовується?
- 11) Що таке оператор введення та для чого він використовується?
- 12) Що таке маніпулятор та для чого він використовується?
- 13) Що таке простір імен?
- 14) Що таке оператор області видимості та для чого він використовується?
- 15) Що таке ініціалізація?

Комп'ютерний практикум №2

Оператор if та логічні оператори

Мета: ознайомитись з використанням *математичних функцій* в C++, вивчити роботу *умовного оператора if*, розглянути можливості застосування *логічних операторів* та *операторів відношення*.

Завдання: створити консольний додаток для реалізації обчислень поставленої задачі згідно отриманого варіанту завдання з використанням *умовного оператора if* та *математичних функцій*.

Загальні вимоги.

- 1) Створити консольний додаток з ім'ям виду *PrizvischeKP2*.
- 2) Програмний код модуля має бути чітко структурований.
- 3) Імена об'єктів мають нести сенсові навантаження.
- 4) Програмний код має супроводжуватись коментарями в тексті програми.

Вимоги до виконання.

- 1) Розробити алгоритм і програму для визначення наведеної залежності при певних значеннях вхідної змінної.
- 2) Вибір залежності для розрахунку вихідної змінної організувати з використанням *умовного оператора if*.
- 3) Виведення результатів розрахунку організувати з вказуванням вхідних значень змінних, при яких був отриманий результат, а також вказуванням варіанту розрахункової залежності, за якою було отримане значення вихідної змінної.
- 4) Підібрати значення вхідних змінних таким чином, щоб отримати значення вихідної змінної за кожним варіантом розрахункової залежності.

Теоретичні відомості

Тема 2.1. Вирази та оператори

2.1.1 Унарні та парні оператори

Існують *унарні оператори* (unary operator) та *парні оператори* (binary operator). Унарні оператори, такі як *оператор звернення до адреси &* та *оператор звернення до значення **, діють на один операнд. Парні чи бінарні оператори, такі як *рівність (==)* та *множення (*)*, діють на два операнди. Деякі

символи, наприклад `*`, використовуються для позначення як унарних (звернення до значення), так і парних (множення) операторів. Тип оператора визначає контекст, в якому він використовується. У використанні цих операторів немає ніякого зв'язку, тому їх слід вважати двома різними символами.

2.1.2 Арифметичні оператори

У мові C/C++ підтримуються наступні арифметичні операції:

- `+` – додавання;
- `-` – віднімання;
- `*` – множення;
- `/` – ділення;
- `%` – остача від ділення.

Усі ці операції є *бінарними (парними)*. Це означає, що для отримання результату, потрібно 2 операнди. Загальний вигляд арифметичної операції:

операнд1 **операція** операнд2

де операція – одна з операцій `+`, `-`, `*`, `%`, `/`.

Операції додавання `+` та віднімання `-` можуть бути як бінарними, так і унарними. Бінарні операції `+` та `-` використовуються у виразах при проведенні обчислень. Унарні операції `+` та `-` використовуються для позначення знаку числа (додатне число або від'ємне число).

```
int a, b;  
a = -8; // унарна операція '-', позначає знак числа  
b = +9; // унарна операція '+', b = 9  
a = b - 5; // бінарна операція '-', використовується у виразі для обчислення
```

2.1.3 Особливості використання операції `%` та `/`

Остачею від ділення націло натурального числа m на натуральне число n є таке ціле число $p < n$, для якого справджується рівність

$$m = kn + p,$$

де k — певне натуральне число, яке називається часткою.

Наприклад, число 1 є остачею від ділення числа 7 на 2, оскільки

$$7 = 3 \times 2 + 1.$$

Якщо число $m < n$, тоді остачею від ділення націло натурального числа m на натуральне число n буде саме число m , оскільки

$$m = 0 \cdot n + p.$$

Наприклад, число 2 є остачею від ділення числа 2 на 7, оскільки

$$2 = 0 \times 7 + 2.$$

Якщо остача від ділення числа m на число n дорівнює нулю, то говорять, що число m ділиться на n без остачі, або, що число m кратне числу n .

// Операція % - взяття остачі від ділення

```
int a, b;  
int c;  
a = 3;  
b = 5;  
c = a % b; // c = 3  
a = 8;  
b = 4;  
c = a % b; // c = 0  
c = 12 % 35; // c = 12  
c = 35 % 12; // c = 11  
c = 16 % 42; // c = 16  
c = 78 % 154; // c = 78  
c = 7 % 4; // c = 3  
c = -5 % -3; // c = -2
```

Операція ділення / має свої особливості, які полягають в наступному:

- якщо два операнди мають цілочисельний тип, то результат повертається цілого типу. У цьому випадку відбувається ділення націло. Остача від ділення відкидається;
- якщо один з операндів має тип з плаваючою комою, тоді результат має також тип з плаваючою комою.

// Операція ділення

```
int a, b;  
int c;  
float x;  
a = 8;  
b = 3;  
c = a / b; // c = 2  
x = a / b; // x = 2.0  
x = a / (float)b; // x = 2.666667  
x = 17.0 / 3; // x = 5.666667  
x = 17 / 3; // x = 5.0
```


2.1.4 Пріоритет арифметичних операторів

В табл. 2.1.1 оператори згруповані за пріоритетом. Унарні арифметичні оператори мають більш високий пріоритет, ніж оператори множення та ділення, які в свою чергу мають більш високий пріоритет, ніж парні оператори віднімання та додавання.

Таблиця 2.1.1 – Арифметичні оператори

Оператор	Дія	Застосування
+	унарний плюс	+ вираз
-	унарний мінус	- вираз
*	множення	вираз * вираз
/	ділення	вираз / вираз
%	залишок	вираз % вираз
+	додавання	вираз + вираз
-	віднімання	вираз - вираз

2.1.5 Логічні оператори та оператори відношення

Загальним результатом *оператора логічного AND (&&)* буде true, якщо обидва його операнди розглядаються як true. *Оператор логічного OR (||)* повертає значення true, якщо будь-який з його операндів розглядається як true.

Оператор логічного NOT (!) повертає обернене значення свого операнда (!true повертає false).

Перелік логічних операторів та операторів відношення, які використовуються в C++, наведені в табл. 2.1.2.

Таблиця 2.1.2 – Логічні оператори та оператори відношення

Оператор	Дія	Застосування
!	логічне NOT	! вираз
<	менше	вираз < вираз
<=	менше чи дорівнює	вираз <= вираз
>	більше	вираз > вираз
>=	більше чи дорівнює	вираз >= вираз
==	рівність	вираз == вираз
!=	не дорівнює	вираз != вираз
&&	логічне AND	вираз && вираз
	логічне OR	вираз вираз

У якості *оператора присвоєння* (assignment operator) в C++ використовується оператор `=` (наприклад, `a=5`) на відміну від логічного *оператора рівності* (equality operator) `==` (перевірка рівності лівої та правої частини виразу в умові), який використовується в логічних виразах (наприклад, `if (a==5) ++i;`).

2.1.6 Складені оператори присвоєння

В C++ також використовують *складені оператори присвоєння* (compound assignment), які існують для кожного з арифметичних операторів (табл. 2.1.3). Ці оператори є зручними, коли в програмі використовуються довгі імена змінних. У цьому випадку відпадає необхідність зайвий раз вводити довге ім'я змінної.

Таблиця 2.1.3 – Складені оператори присвоєння

Оператор	Застосування	Приклад
<code>*=</code>	вираз <code>*=</code> вираз	<code>a *= b</code> рівноцінно <code>a = a * b</code>
<code>/=</code>	вираз <code>/=</code> вираз	<code>a /= b</code> рівноцінно <code>a = a / b</code>
<code>%=</code>	вираз <code>%=</code> вираз	<code>a %= b</code> рівноцінно <code>a = a % b</code>
<code>+=</code>	вираз <code>+=</code> вираз	<code>a += b</code> рівноцінно <code>a = a + b</code>
<code>-=</code>	вираз <code>-=</code> вираз	<code>a -= b</code> рівноцінно <code>a = a - b</code>

2.1.7 Оператори інкременту та декременту

В C++ існують *оператор прирощення* `++` (increment) та *оператор зменшення* `--` (decrement), які дозволяють в короткій та зручній формі додавати чи віднімати одиницю з об'єкту. Ця форма запису часто використовується при роботі з *ітераторами*, оскільки більшість ітераторів не підтримують арифметичні дії.

Ці оператори існують в двох формах: префіксній (prefix) та постфіксній (postfix). *Префіксний оператор прирощення (зменшення)* здійснює збільшення (зменшення) свого операнду на одиницю і повертає *змінений* об'єкт як результат. *Постфіксний оператор прирощення (зменшення)* повертає копію вихідного операнду *незмінною*, а потім здійснює збільшення (зменшення) свого операнду на одиницю.

```
int i = 0, j;
j = ++i;
std::cout << "j=" << j << ", i=" << i << std::endl;
// j=1, i=1 префікс повертає збільшене значення
int k = 0, l;
l = k++;
std::cout << "l=" << l << ", k=" << k << std::endl;
// l=0, k=1 постфікс повертає вихідне значення
```


2.1.8 Оголошення using

До сих пір імена зі стандартної бібліотеки згадувались у програмах явно, тобто перед кожним з них було вказано ім'я простору імен `std`. Наприклад, при читанні зі стандартного пристрою введення застосовувалась форма запису `std::cin`. Тут застосований оператор області видимості `::`. При частому використанні бібліотечних імен така форма запису може бути надто громіздкою. Щоб уникнути цього застосовують оголошення `using` (`using declaration`). Оголошення `using` дозволяє використовувати імена з іншого простору імен без вказування префіксу `ім'я_простору_імен::`. Оголошення `using` має наступний формат:

```
using простір_імен::ім'я;
```

Після того як оголошення `using` було зроблено один раз, до вказаного в ньому імені можна звертатися без вказування простору імен.

```
#include <iostream>
using std::cout;
using std::endl;
int main()
{
    int i = 0, j;
    j = ++i;
    cout << "j=" << j << ", i=" << i << endl;
    return 0;
}
```

Для кожного імені необхідно індивідуальне оголошення `using`. Оголошення `using` не можна застосовувати у файлах заголовку для запобігання конфлікту імен.

2.1.9 Використання математичних функцій

Щоб використовувати у програмі математичні функції необхідно приєднати до програми заголовковий файл `math.h` за допомогою директиви препроцесора `#include`.

```
#include <math.h>
```

Перелік основних математичних функцій, які присутні у бібліотеці `math`, наведений у таблиці 2.1.4.

Таблиця 2.1.4 – Основні математичні функції

Функція	Пояснення
abs(x)	модуль цілого числа
fabs(x)	модуль дробового числа
sin(x)	синус
cos(x)	косинус
tan(x)	тангенс
asin(x)	арксинус
acos(x)	арккосинус
atan(x)	арктангенс
log(x)	натуральний логарифм
log10(x)	десятковий логарифм
exp(x)	піднесення e до ступеню x
pow(x,y)	піднесення x до ступеню y
pow10(x)	піднесення 10 до ступеню x
sqrt(x)	квадратний корінь
ceil(x)	округлення вгору
floor(x)	округлення вниз
fmod(x,y)	остача від ділення x на y

Тема 2.2. Засоби керування ходом виконання програми

2.2.1 Оператори керування потоком

Оператори (statement) – це програмні елементи, які контролюють як і в якій послідовності виконуватимуться вирази. В мові C++ визначений набір операторів *керування потоком* (flow of control), які забезпечують більш складні шляхи виконання коду ніж послідовне.

Більшість операторів в мові C++ закінчуються крапкою з комою. Вираз типу `ival + 5` стає *оператором виразу* (expression statement), який завершується крапкою з комою. Як правило, вирази містять оператори, результат обчислень яких впливає на стан програми. До таких операторів відносяться присвоєння, інкремент, введення та виведення.

Сама проста форма оператора – це *пустий* (empty), або *нульовий оператор* (null statement). Він представляє собою символ крапки з комою (;). Пустий оператор використовують у випадку, коли синтаксис мови потребує наявності оператора, а логіка програми – ні. Зазвичай це відбувається у випадку, коли вся робота циклу здійснюється в його умові. Наприклад, можна організувати введення, ігноруючи всі прочитані дані, поки не зустрінеться певне значення:


```
//цикл виконується доки не зустрінеться кінець файлу
//або значення рівне змісту змінної sought
while (cin >> s && s != sought)
    ; //пустий оператор
```

В більшості випадків пустий оператор безпечний, але помилкова крапка з комою в кінці умови циклу `while` або оператора `if` може призвести до помилкового коду:

```
vector<string> svec;
vector<string>::iterator iter;
while (iter != svec.end()); //тіло циклу пусте!
    ++iter; // оператор інкременту не є частиною циклу
```

Складений оператор (compound statement), зазвичай називають *блоком* (block), він являє послідовність операторів, які заключені у *фігурні дужки*. Об'єднавши кілька операторів, узявши їх у фігурні дужки, можна отримати блок операторів, який розглядається як один оператор. Блок операторів має власну *область видимості*. Об'явлені у блоці імена доступні тільки у даному блоці і блоках, вкладених у нього. Як зазвичай, ім'я видиме тільки з моменту коли воно визначене і до кінця блоку включно. В кінці блоку не ставиться крапка з комою.

Змінні можна визначати в керуючих структурах операторів `if`, `switch`, `while` та `for`. Змінні, які визначені у керуючій структурі, видимі лише в межах цього оператора. Якщо до змінної потрібно звертатися після роботи керуючого оператора, то її слід визначити поза межами оператора.

2.2.2 Оператори умовного виконання

Мова C++ включає два оператори, які забезпечують умовне виконання. *Оператор if* розділяє потік виконання на основі умови. *Оператор switch* обраховує результат цілочисельного виразу і на його основі обирає один з кількох шляхів виконання.

2.2.2.1 Оператор if

Оператор if який забезпечує виконання операторів за умовою. Структуру оператора можна представити наступним чином:

```
if (true) //умова виконання блоку операторів
{
    //блок операторів 1
}
else //якщо умова не виконана
```



```
{  
    //блок операторів 2  
}
```

Існують дві форми оператора `if`: з розділом `else` та без нього. В обох версіях умова береться у круглі дужки. Умова може бути виразом або ініціалізуючим оголошенням змінної. Тип виразу або змінної повинен дозволяти перетворення в тип `bool`. Оператор `if` – керуючий оператор, який забезпечує виконання на основі значення певної умови. Якщо умова істинна (значення `true`), виконується тіло оператора `if`. В іншому випадку (значення `false`) управління переходить до розділу `else` (якщо він існує).

Оператор `if` слугує для того, щоб виконати будь-яку операцію в тому випадку, коли умова є вірною. Умовна конструкція в C++ завжди записується в круглих дужках після оператора `if`.

У середині фігурних дужок вказується тіло умови. Якщо умова виконається, то почнеться виконання всіх команд, які знаходяться між фігурними дужками.

```
int число = 5;  
if (число < 10) { // Якщо введене число менше 10.  
    std::cout << "Це число менше 10." << std::endl;  
}  
else { // в іншому випадку  
    std::cout << "Це число більше або дорівнює 10." << std::endl;  
}
```

В прикладі перевіряється умова чи `число < 10`, після чого виконується блок коду у фігурних дужках. Якщо умова `false`, то виконується блок коду у фігурних дужках після оператору `else`.

Якщо необхідно перевірити більше однієї умови, використовується поєднання операторів `if` та `else`.

Кожному оператору `if` відповідає лише один оператор `else`. Сукупність цих операторів - `else if` означає, що якщо не виконалась попередня умова, то потрібно перевірити дану умову. Якщо жодна з умов не вірна, то виконується тіло оператора `else`.

```
int число = 5;  
if (число < 10) { // Якщо введене число менше 10.  
    std::cout << "Це число менше 10." << std::endl;  
}  
else if (число == 10) { // в іншому випадку перевірити наступну умову  
    std::cout << "Це число дорівнює 10." << std::endl;  
}  
else { // в іншому випадку  
    std::cout << "Це число більше 10." << std::endl;  
}
```



```
}
```

Зверніть увагу на умову `else if (число == 10)`. В цій умові рівність числа десяти перевіряється за допомогою оператора `==`. Якщо випадково замість оператора тотожності `==` в умові записати `=`, а це оператор присвоєння в C++, то змінній буде присвоєне нове значення і результатом перевірки умови завжди буде `true`.

Якщо після оператора `if`, `else` або їх зв'язки `else if` повинна виконуватися тільки одна команда, то фігурні дужки можна не ставити. Попередню програму можна записати в такий спосіб:

```
int число = 5;
if (число < 10)    // Якщо введене число менше 10.
    std::cout << "Це число менше 10." << std::endl;
else if (число == 10) // в іншому випадку перевірити наступну умову
    std::cout << "Це число дорівнює 10." << std::endl;
else // в іншому випадку
    std::cout << "Це число більше 10." << std::endl;
```

В даному випадку потрібно розуміти, що оператори `if`, `else if` та `else` записуються в один рядок. Ознакою завершення рядка є оператор `;`. В більш спрощеній формі даний код матиме вигляд:

```
int число = 5;
if (число < 10) std::cout << "Це число менше 10." << std::endl;
else if (число == 10) std::cout << "Це число дорівнює 10." << std::endl;
else std::cout << "Це число більше 10." << std::endl;
```

2.2.2.2 Втрачений оператор `else`

При такій формі запису може трапитись помилка *втраченого оператора `else`* (*dangling else*). Ця помилка виникає коли не використовуються блоки з фігурних дужок після операторів `if` та `else` та кількість операторів `if` більша ніж операторів `else`. Кожен оператор `else` відноситься до найближчого оператора `if`.

Приклад такої помилки наведений нижче. Нам потрібно перевірити умову і якщо вона виконується, то перевірити наступну умову – чи число більше 10:

```
int число = 5;
bool умова = 1;
if (умова) // Якщо умова істинна
    std::cout << "Умова виконана." << std::endl;
    if (число > 10) // Перевіряємо чи число > 10
        std::cout << "Це число більше 10." << std::endl;
else // в іншому випадку
    std::cout << "Умова не виконана." << std::endl;
```


Результатом роботи даного коду буде наступний результат:

Умова виконана.

Умова не виконана.

Щоб виправити цю помилку потрібно застосувати фігурні дужки.

```
int число = 5;
bool умова = 1;
if (умова) { // Якщо умова істинна
    std::cout << "Умова виконана." << std::endl;
    if (число > 10) // Перевіряємо чи число > 10
        std::cout << "Це число більше 10." << std::endl;
}
else // в іншому випадку
    std::cout << "Умова не виконана." << std::endl;
```

Після цього отримаємо очікуваний результат:

Умова виконана.

2.2.2.3 Вкладений оператор if

В попередньому прикладі був наведений випадок використання *вкладеного оператора if*, тобто коли один оператор *if* знаходиться всередині іншого оператора *if*. При використанні вкладених операторів *if* обов'язковим є застосування фігурних дужок для чіткого розподілення на блоки.

Також фігурні дужки є необхідними при потребі виконання більше ніж однієї команди після перевірки умови.

```
#include "stdafx.h"
#include <iostream>

int main()
{
    setlocale(0, ""); // Для виведення кирилиці
    int число = 5;
    int прапорець;
    if (число < 10) { // Якщо введене число менше 10.
        std::cout << "Це число менше 10." << std::endl;
        прапорець = 1;
    }
    else if (число == 10) { // інакше перевірити наступну умову
        std::cout << "Це число дорівнює 10." << std::endl;
        прапорець = 2;
    }
}
```



```
else { // в іншому випадку
    std::cout << "Це число більше 10." << std::endl;
    прапорець = 3;
}
return 0;
}
```

Якщо виконується перша умова в операторі `if`, то виконується відповідний блок операторів, після чого робота умовного оператора закінчується. Якщо умова хибна, виконання переходить до наступного оператора `else if` і перевіряється його умова, при виконанні якої запускається відповідний цій умові блок операторів і робота умовного оператора закінчується. Якщо не виконується і ця умова, виконання переходить до блоку `else` та виконується вкладений в нього блок операторів.



Приклад програмної реалізації

Приклад роботи з умовним оператором if та математичними функціями.

// ConsoleApplication44.cpp : Програма розрахунку вихідної змінної у залежності від значення вхідної із використанням умовного оператора.
//

```
#include "stdafx.h"  
#include <windows.h> //Необхідно для виведення в консолі української мови  
#include <math.h> //Бібліотека математичних функцій  
#include <iostream>  
using std::cout;  
using std::cin;  
using std::endl;
```

```
int main()  
{  
    SetConsoleCP(1251); //Необхідно для виведення в консолі  
                        //української мови  
    SetConsoleOutputCP(1251); //Необхідно для виведення в консолі  
                              //української мови  
  
    double t, x, z;  
    unsigned i; //беззнакове ціле число  
    cout << "Задайте значення вихідної змінної t" << endl;  
    cin >> t;  
    x = exp(-sin(t)) + cos(t);  
    if (x<=-3)  
    {  
        z = sqrt(abs(x))/(t+x);  
        i = 1;  
    }  
    else if (x > -3 && x<2)  
    {  
        z = t+tan(pow(x,3));  
        i = 2;  
    }  
    else if (x==2)  
    {  
        z = sin(pow(x, 3));  
        i = 3;  
    }  
    else  
    {  
        z = log(x + t);  
        i = 4;  
    }  
    cout << "При t=" << t << " значення x=" << x << ", розрахунок
```



```
    проведений за залежністю " << i << ", z=" << z << endl;  
    return 0;  
}
```

C++

Завдання до комп'ютерного практикуму №2

1. Розробити алгоритм і програму для визначення залежності:

$$y = \begin{cases} |\sin x^2 - b \cdot \cos^2 x^3| \text{ при } x < 0 \\ \lg|x^2 - 5,1| + l^{-x^2+1} \text{ при } 0 \leq x < 1 \\ tg^2(x-2) - 2,5 \text{ при } x \geq 1 \end{cases}$$

де $x = a * \sin^2 t - b * \cos t^3$; $a = 1,3$; $b = 1,5$; $t = 1,8$.

2. Розробити алгоритм і програму для визначення залежності:

$$y = \begin{cases} \ln|(x-1,5) \cdot \sin x| + 2 \cdot ax^2 & \text{при } x < 1 \\ tgx - b \cdot \cos^2 x^3 & \text{при } 1 \leq x \leq 2,1 \\ l^{-x^2+1,5x} + \ln x & \text{при } x > 2,1 \end{cases}$$

де $a = 1,2$; $b = 2,1$; $x = a * \ln t$; $t = 2,8$.

3. Розробити алгоритм і програму для визначення залежності:

$$y = \begin{cases} \ln|x-3| + 1,4 & \text{при } x < 0 \\ tg(x-1) + a \cdot \cos^2 x^3 & \text{при } 0 \leq x < 1 \\ \ln(x^2 + 1) - e^{-2x} & \text{при } x \geq 1 \end{cases}$$

де $x = a * \sin t - b * \cos(2t)$; $a = 2,5$; $b = 0,8$; $t = 2,93$.

4. Розробити алгоритм і програму для визначення залежності:

$$Z = \begin{cases} 3 & \text{при } x = 1,5 \\ \ln x & \text{при } x > 2 \\ \sqrt{|x|} & \text{при } x \leq -3 \\ |x + tgx^2| & \text{при } -3 < x \leq 2 \end{cases}$$

де $x = e^{-\sin t} + \cos t$, $t = 1,37$.

5. Розробити алгоритм і програму для визначення залежності:

$$Z = \begin{cases} e^{-x} + tgx & \text{при } 2 \leq x \leq 2,5 \\ \ln x^2 + b \cdot \ln x^4 & \text{при } x > 2 \\ \cos^2 x + b \cdot \cos^3 x & \text{при } x < 2 \end{cases}$$

де $x = a * \sin^3 t + 1,5$; $a = 1,2$; $b = 2,3$; $t = 0,8$.

6. Розробити алгоритм і програму для визначення залежності:

$$Y = \begin{cases} \ln|\sin^3 x^3| - 1 & \text{при } x \leq 2 \\ 10^{-2x^2 + \sin x} & \text{при } 2 < x \leq 2,3 \\ \cos x + 2\sin x^2 & \text{при } x > 2,3 \end{cases}$$

де $x = a * t - \text{ctg}(b * t)$; $a = 0,2$; $b = -0,4$; $t = 4,7$.

7. Розробити алгоритм і програму для визначення залежності:

$$Y = \begin{cases} \lg(x^2 - 2x) & \text{при } x < 0 \\ \ln(x + 4) & \text{при } 0 \leq x < 10 \\ \log_x 5 - \log_5 x & \text{при } x \geq 10 \end{cases}$$

де $x = a * \cos^2 t - b * t$; $t = 1$; $a = 0,5$; $b = 6$.

8. Розробити алгоритм і програму для визначення залежності:

$$Y = \begin{cases} |tgx - \cos^3 2x| & \text{при } x < 0 \\ x^2 + x^3 - 5x^4 & \text{при } 0 \leq x < 5 \\ x + \lg x & \text{при } x > 5 \end{cases}$$

де $x = a * t + b$; $a = 5$; $t = 4$; $b = 0,1$.

9. Розробити алгоритм і програму для визначення залежності:

$$Y = \begin{cases} |x^2 - 3x| & \text{при } x < 0 \\ t^2 - 4 & \text{при } x = 0 \\ -x^3 - x & \text{при } x > 0 \end{cases}$$

де $x = \exp(t^4 + t + 2t - t)$; $t = 4$.

10. Розробити алгоритм і програму для визначення залежності:

$$Y = \begin{cases} |\sin x^2 - \cos^2 x| & \text{при } x < 1 \\ \cos 2x + \sin 2x & \text{при } 1 \leq x < 2 \\ |x - 5| & \text{при } x \geq 2 \end{cases}$$

де $x = t^2 + a * \cos t$; $a = 2$; $t = 0$.

11. Розробити алгоритм і програму для визначення залежності:

$$Y = \begin{cases} |\log_7 7x| & \text{при } x < 0 \\ 5x + 4 & \text{при } 0 \leq x < 15 \\ -x^{\sin x + 3} & \text{при } x \geq 15 \end{cases}$$

де $x = t^{\frac{a}{b}}$; $t = 4$; $a = 3$; $b = 2$.

12. Розробити алгоритм і програму для визначення залежності:

$$Y = \begin{cases} e^{x^2} & \text{при } x < -3 \\ |x| & \text{при } -3 \leq x < 3 \\ \sin x - \cos 2x & \text{при } x \geq 3 \end{cases}$$

де $x = t^3 - a * t + b$; $a = 2$; $b = 4$; $t = 1$.

13. Розробити алгоритм і програму для визначення залежності:

$$Y = \begin{cases} -e^{-x^2+2x} & \text{при } x < 3 \\ \sin x^4 + \operatorname{ctg} 2x & \text{при } 3 \leq x < 4 \\ 5 & \text{при } x \geq 4 \end{cases}$$

де $x = \operatorname{Cos}(\ln t^a + b)$; $a = 0,3$; $b = 2,1$; $t = 1,7$.

14. Розробити алгоритм і програму для визначення залежності:

$$Y = \begin{cases} \sin|x| & \text{при } x < 10 \\ 10^x + \lg 3x^2 & \text{при } x = 10 \\ \cos|x| & \text{при } x > 10 \end{cases}$$

де $x = a * \sin t^b - \cos(a * t)$; $a = 2$; $b = 3$; $t = 1,5$.

15. Розробити алгоритм і програму для визначення залежності:

$$Y = \begin{cases} \operatorname{tg}^2|x^2 - 4x + 3| & \text{при } x < 2 \\ |2x + 5| & \text{при } 2 \leq x < 5 \\ 4 & \text{при } x \geq 5 \end{cases}$$

де $x = 4t^3 - a * t + t^{-b/a}$; $a = 1,5$; $b = -2$; $t = -0,5$.

16. Розробити алгоритм і програму для визначення залежності:

$$Y = \begin{cases} \cos^2(x + 4) & \text{при } x > 2 \\ |\sin x - e^x| & \text{при } 2 \leq x < 4 \\ \ln 3x & \text{при } x \geq 4 \end{cases}$$

де $x = a * t^3 - b * t$; $a = 1,5$; $b = 3$; $t = 2,3$.

17. Розробити алгоритм і програму для визначення залежності:

$$Y = \begin{cases} e^{-x} + \operatorname{ctgx} \cdot \operatorname{tg} 2x & \text{при } x < -10 \\ |x^3| & \text{при } -10 \leq x < -1 \\ x^4 + 10^x & \text{при } x \geq -1 \end{cases}$$

де $x = a * t - b * t^2$; $a = 1,5$; $b = 3$; $t = 2,3$.

18. Розробити алгоритм і програму для визначення залежності:

$$Y = \begin{cases} \sin^4(x-5) & \text{при } x < 0 \\ x^3 - \sqrt{x} & \text{при } 0 \leq x < 10 \\ e^{-x} + |x| & \text{при } x \geq 10 \end{cases}$$

де $x = \frac{\tan t}{\sin t + \cos t}$; $t = 1,2$.

19. Розробити алгоритм і програму для визначення залежності:

$$Y = \begin{cases} \frac{1}{x} + \frac{1}{3x^2} & \text{при } x < 5 \\ 7x^{\ln x} & \text{при } 5 \leq x < 7 \\ 2 \sin x^4 + \cos x & \text{при } x \geq 7 \end{cases}$$

де $x = a * \cos t^2 - b * t$; $a = 1$; $b = 0,2$; $t = 0,4$.

20. Розробити алгоритм і програму для визначення залежності:

$$Y = \begin{cases} e^{-2x} + x & \text{при } x < 0 \\ x^4 + 5x & \text{при } x = 0 \\ |\sin x + \operatorname{tg} x| & \text{при } x > 0 \end{cases}$$

де $x = b * t^2 + a$; $a = -3$; $b = 4$; $t = 3,5$.

21. Розробити алгоритм і програму для визначення залежності:

$$Y = \begin{cases} 5^{\operatorname{tg} x} - 4 & \text{при } x < 2 \\ \frac{1}{2 \sin x + 3} & \text{при } 2 \leq x < 3 \\ (2 \sin x)^x & \text{при } x \geq 3 \end{cases}$$

де $x = 2,5t^2 - a / t$; $a = 5$; $t = 7,8$.

22. Розробити алгоритм і програму для визначення залежності:

$$Y = \begin{cases} \cos 5x - x^{2,3} & \text{при } x < -2 \\ |\ln x^4 - 5| & \text{при } -2 \leq x < 2 \\ \frac{1}{\operatorname{tg} x + \operatorname{ctg} 2x} & \text{при } x \geq 2 \end{cases}$$

де $x = a * \sin t^b - a$; $a = 0,2$; $b = 0,4$; $t = 0,3$.

23. Розробити алгоритм і програму для визначення залежності:

$$Y = \begin{cases} 10^{\sin x - \cos 2x} & \text{при } x < 3 \\ 5 & \text{при } 3 \leq x < 9 \\ |x^6 - 7\sqrt{x}| & \text{при } x \geq 9 \end{cases}$$

де $x = \ln t^a + b$; $a = 3$; $b = 0,3$; $t = 2,7$.

24. Розробити алгоритм і програму для визначення залежності:

$$Y = \begin{cases} x^2 + 4 & \text{при } x < -2 \\ |\sin x - 2^{-x}| & \text{при } -2 \leq x < 10 \\ \log_5 x^2 & \text{при } x \geq 10 \end{cases}$$

де $x = t^a + b * t^b$; $a = 0,3$; $b = 0,4$; $t = 0,2$.

25. Розробити алгоритм і програму для визначення залежності:

$$Y = \begin{cases} \lg(3x - x^2) & \text{при } 0 < x < 2 \\ \sin(3x^3 + 4) & \text{при } 2 \leq x < 3 \\ \frac{1}{x^4 + 5} & \text{при } x \geq 3 \end{cases}$$

де $x = t^{a * b - b}$; $a = 0,3$; $b = 1,1$; $t = 0,7$.

26. Розробити алгоритм і програму для визначення залежності:

$$Y = \begin{cases} |\sin^2 x + \cos^3 2x| & \text{при } x < 2 \\ \frac{x^3 - 5x^4}{x^2} & \text{при } 2 \leq x < 4 \\ \frac{a \cdot x + \lg x}{b} & \text{при } x \geq 4 \end{cases}$$

де $x = a * t + b$; $a = 1,8$; $t = 2$; $b = 3,4$.

27. Розробити алгоритм і програму для визначення залежності:

$$Y = \begin{cases} |\sin^2 x - \cos x^2| & \text{при } x < 0,5 \\ \cos a \cdot x + \sin a \cdot x & \text{при } 0,5 \leq x < 3,1 \\ |x - a| & \text{при } x \geq 3,1 \end{cases}$$

де $x = t^2 + a \cdot \sin t$; $a = 2$; $t = 0.5$.

28. Розробити алгоритм і програму для визначення залежності:

$$Y = \begin{cases} e^{-bx} + tg^2 x / x & \text{при } 1,5 < x < 3,5 \\ \lg x^2 + b \cdot \lg x^4 & \text{при } x > 3,5 \\ \cos^3 x + b \cdot \cos a \cdot x & \text{при } x < 1,5 \\ t \cdot \sin b \cdot x / a & \text{при } x = 3,5 \end{cases}$$

де $x = a \cdot \sin^2 t + 1.9$; $a = 2.7$; $b = 1.5$; $t = 0.8$.

29. Розробити алгоритм і програму для визначення залежності:

$$Y = \begin{cases} \lg(x - x^3) & \text{при } 0 < x < 5 \\ \cos(2x^3 + 3) & \text{при } 5 \leq x < 7 \\ \frac{1}{x^2 + 3} & \text{при } x \geq 7 \end{cases}$$

де $x = t^{a \cdot b - b}$; $a = 0,5$; $b = 1,5$; $t = 0,9$.

30. Розробити алгоритм і програму для визначення залежності:

$$Y = \begin{cases} e^{x^2 + 3x} & \text{при } x < 3 \\ \cos x^3 + tg 3x & \text{при } 3 \leq x < 4 \\ \sin 5x & \text{при } x \geq 4 \end{cases}$$

де $x = \sin(\ln t^a + b)$; $a = 0,2$; $b = 2,5$; $t = 1,6$.

Контрольні питання

- 1) Опишіть структуру та принцип роботи оператора `if`?
- 2) Унарні та прані оператори. У чому їх відмінність?
- 3) Перерахуйте арифметичні оператори згідно з їх пріоритетом.
- 4) Перерахуйте логічні оператори та оператори відношення.
- 5) Перерахуйте складені оператори присвоєння та поясніть принцип їх дії.
- 6) Що таке оператор прирощення та оператор зменшення? Поясніть їх використання на прикладі.
- 7) Що таке префіксний оператор та постфіксний оператор? Поясніть відмінність між ними.
- 8) Що таке оголошення `using` та для чого воно використовується?
- 9) Що потрібно для використання у програмі математичних функцій? Перерахуйте основні математичні функції.



Комп'ютерний практикум №3

Засоби керування процесом виконання програми

Мета: ознайомитись із операторами для зміни послідовності виконання програми, вивчити роботу оператора *ітераційного циклу while*, вивчити роботу оператора *арифметичного циклу for*, вивчити роботу генератора випадкових чисел.

Завдання: створити консольний додаток для реалізації обчислень поставленої задачі згідно двох частин отриманого варіанту завдання з використанням операторів циклу *while* та *for*, а також умовного оператора *if*.

Загальні вимоги.

- 1) Створити консольний додаток з ім'ям виду *PrizvischeKP3*.
- 2) Програмний код модуля має бути чітко структурований.
- 3) Імена об'єктів мають нести сенсові навантаження.
- 4) Програмний код має супроводжуватись коментарями в тексті програми.

Вимоги до виконання.

Частина 1.

- 1) Створити програму для розрахунку шуканих значень результуючої величини в заданому діапазоні значень вхідної змінної.
- 2) При програмуванні використовувати *арифметичний цикл for*.
- 3) Вхідні константні значення об'являти у вигляді *константних змінних* певного типу.
- 4) Виведення результатів розрахунку організувати у табличному вигляді з виведенням вхідної та вихідної змінної на кожній ітерації розрахунку.

Частина 2.

- 1) Створити програму для розрахунку наближеного значення функції за допомогою суми нескінченного ряду.
- 2) Організувати перевірку належності введеного значення змінної *x* вказаному інтервалу за допомогою *умовного оператора if*.
- 3) Організувати розрахунок суми нескінченного ряду з використанням *ітераційного оператора while*. Умовою виходу з циклу має бути точність обчислень ряду.

- 4) Передбачити виведення на екран значень суми на кожній ітерації розрахунку. Вивести на екран контрольне значення розрахованої функції.

Теоретичні відомості

Тема 3.1. Ітераційні оператори

Ітераційні оператори (iterative statement), називають також *циклами* (loop). Вони забезпечують повторне виконання коду, доки їх умова істинна. Оператори while та for перевіряють умову до виконання тіла циклу (*цикли з передумовою*). Оператор do while спочатку виконує тіло циклу, а потім перевіряє свою умову (*цикл з післяумовою*).

3.1.1 Оператор while

Оператор *while* організовує ітераційне (циклічне) виконання фрагменту коду, поки його умова залишається істинною. Структура оператору while виглядає наступним чином

```
while (true) //умова
{
    //блок операторів
}
```

Оператор while (оператор ітераційного циклу) циклічно виконує *блок операторів*, доки *умова* залишається істинною. *Умова* – це вираз, результатом виконання якого є істина (true) чи брехня (false). Допоки *умова* істинна (true), *блок операторів* виконується. Після виконання *блоку операторів* умова перевіряється знову. Якщо *умова* залишається істинною, *блок операторів* виконується знову. Цикл while продовжується, по чергово перевіряючи *умову* та виконуючи *блок операторів*, допоки *умова* не стане брехнею (false).

Блок (block) – це послідовність з будь-якої кількості операторів, що заключено у фігурні дужки. Блок є оператором і може використовуватись будь-де, де припустимий один оператор.

```
while (змінна<=10) //цикл виконується допоки змінна не перевищить 10
{
    сума += змінна; //додати до сума значення змінна
    ++змінна; //додати 1 до змінна
}
```

Нижче наведений приклад використання циклу while, в якому рахується сума чисел від 1 до 1000.


```
#include "stdafx.h"
#include <iostream>
using namespace std;

int main()
{
    setlocale(0, ""); // Встановлення кирилиці в консолі
    int лічильник = 0; // Ініціалізуємо лічильник
    int сума = 0; // Ініціалізуємо значення суми
    while (лічильник < 1000)
    {
        лічильник++;
        сума += лічильник;
    }
    cout << "Сума чисел від 1 до 1000 = " << сума << endl;
    cout << "Кількість ітерацій = " << лічильник << endl;
    return 0;
}
```

3.1.2 Оператор do while

Оператор do while схожий на оператор while і організовує ітераційне (циклічне) виконання фрагменту коду, поки його умова залишається істинною. Структура оператору while виглядає наступним чином

```
do {
    //блок операторів
} while (true); //умова
```

Відмінність циклу do while від циклу while полягає в тому, що цикл do while є оператором циклу з післяумовою, а цикл while є циклом з передумовою. Це впливає на мінімально можливу кількість виконання циклу. Цикл з передумовою виконується 0 та більше разів, цикл з післяумовою виконується 1 або більше разів. Цикл do while спочатку виконується один раз, а потім перевіряє умову, тобто він виконається мінімум один раз. Цикл while спочатку перевіряє умову і якщо умова не виконується, то тіло циклу не виконається жодного разу.

Попередній приклад з циклом do while виглядатиме наступним чином:

```
#include "stdafx.h"
#include <iostream>
using namespace std;

int main()
{
    setlocale(0, ""); // Встановлення кирилиці в консолі
```

```
int лічильник = 0; // Ініціалізуємо лічильник
int сума = 0; // Ініціалізуємо значення суми
do
{
    лічильник++;
    сума += лічильник;
} while (лічильник < 1000);
cout << "Сума чисел від 1 до 1000 = " << сума << endl;
cout << "Кількість ітерацій = " << лічильник << endl;
return 0;
}
```

В циклі `do while` оператор (блок операторів) виконується перше, ніж перевіриться умова. При цьому умова не може бути пустою. Якщо умова брехня, цикл завершується, в іншому разі цикл повторюється. Змінні, які використовуються в умові, слід визначити за межами циклу `do while`.

Наступний приклад сумує будь-яку кількість пари чисел за допомогою циклу `do while`.

```
#include "stdafx.h"
#include <iostream>
#include <string>
#include <locale>;
#include "windows.h";
using namespace std;

int main()
{
    //Встановлення таблиці кодування для введення укр. мови
    SetConsoleCP(1251);
    //Встановлення таблиці кодування для виведення укр. мови
    SetConsoleOutputCP(1251);
    string продовжити;
    do
    {
        cout << "Введіть два значення:\n";
        int число1 = 0, число2 = 0;
        // Значення вводяться через пробіл або Enter
        cin >> число1 >> число2;
        cout << "Сума " << число1 << " та " << число2 << " = "
             << число1 + число2 << "\n";
        << "Продовжити введення? Введіть 'так' чи 'ні': ";
        cin >> продовжити;
    } while (!продовжити.empty() && продовжити[0] != 'н');
    return 0;
}
```


Цикл починається запитом у користувача двох чисел. Потім виводиться їх сума і іде запит, чи бажає користувач підсумувати далі. Відповідь користувача перевіряється в умові. Якщо введення пуста або починається з н, цикл завершується. В іншому випадку цикл повторюється.

Приклад роботи даної програми наведений нижче:

Введіть два значення:

78

69

Сума 78 та 69 = 147

Продовжити введення? Введіть 'так' чи 'ні': так

Введіть два значення:

49

36

Сума 49 та 36 = 85

Продовжити введення? Введіть 'так' чи 'ні': ні

Press any key to continue . . .

Оскільки умова не обробляється до завершення оператору чи блоку, цикл `do while` не дозволяє визначати змінні в умові. Якщо визначити змінні в умові, то будь-яке їх використання відбудеться до визначення, що викличе помилку.

3.1.3 Оператор `for`

Оператор `for` (оператор арифметичного циклу) організовує задану кількість повторень блоку операторів. У кожного оператора `for` є дві частини: *заголовок* та *тіло*. Заголовок контролює кількість разів виконання *тіла* циклу. Сам заголовок складається з трьох частин: *оператора ініціалізації*, *умови* і *виразу*.

```
for (оператор ініціалізації; умова; вираз) //заголовок циклу
{
    //тіло циклу
}
```

Оператор ініціалізації визначає змінну циклу та задає її початкове значення. В умові міститься перевірка досягнення змінною циклу кінцевого значення. У виразі здійснюється приращення змінної циклу.

```
double сума = 0; // Ініціалізуємо значення суми
double число = 15.6; // Ініціалізуємо значення змінної
for (int i = 0; i < 10; i++) //заголовок циклу
{ //тіло циклу
    сума += число;
    cout << i+1 << ".\tСума = " << сума << endl;
```


}

В даному прикладі в *операторі ініціалізації* визначається *змінна циклу* і цілочисельного типу `int` та ініціалізується значенням `0`. В умові перевіряється чи не досягла *змінна циклу* кінцевого значення `10`. У виразі вказується прирощення *змінної циклу*, яке дорівнює `1`. Якщо після перевірки умови *змінна циклу* не досягла свого кінцевого значення, виконується *тіло циклу*. Після виконання *тіла циклу* відбувається прирощення *змінної циклу* на `1` і знову перевіряється умова досягнення *змінною циклу* кінцевого значення.

В операторі арифметичного циклу `for` на відміну від оператору ітераційного циклу `while` заздалегідь відома кількість *ітерацій* – повторень *тіла циклу*. Вона дорівнює

$$\text{кількість ітерацій} = \frac{\text{кінцеве значення змінної циклу} - \text{початкове значення змінної циклу}}{\text{прирощення змінної циклу}}$$

Для наведеного прикладу кількість ітерацій циклу дорівнюватиме $(10 - 0)/1 = 10$, тобто перед початком роботи оператора `for` відомо, що цикл виконається 10 разів.

Результатом роботи попереднього циклу буде:

1. Сума = 15.6
2. Сума = 31.2
3. Сума = 46.8
4. Сума = 62.4
5. Сума = 78
6. Сума = 93.6
7. Сума = 109.2
8. Сума = 124.8
9. Сума = 140.4
10. Сума = 156

У мові C++, як і в деяких інших існують різні варіації запису циклу `for`. В першу чергу це стосується запису заголовку циклу. Заголовок циклу `for` складається з трьох частин - оператора ініціалізації, умови і виразу, при цьому одна, дві або всі три частини у заголовку можуть бути пропущені. Єдиною обов'язковою умовою є наявність двох операторів `;`, які вказують на необхідність присутності параметру, навіть коли він пропущений.

Далі наведені кілька прикладів циклу `for` з пропущеними параметрами у заголовку циклу.

```
double сума = 0; // Ініціалізуємо значення суми
double число = 15.6; // Ініціалізуємо значення змінної
```



```
int i = 0;
for (; i < 10; i++) //заголовок циклу
{ //тіло циклу
    сума += число;
    cout << i+1 << ".\tСума = " << сума << endl;
}

double сума = 0; // Ініціалізуємо значення суми
double число = 15.6; // Ініціалізуємо значення змінної
int i = 0;
for ( ; i < 10; ) //заголовок циклу
{ //тіло циклу
    сума += число;
    cout << i+1 << ".\tСума = " << сума << endl;
    i++;
}

double сума = 0; // Ініціалізуємо значення суми
double число = 15.6; // Ініціалізуємо значення змінної
int i = 0;
for ( ; ; ) //заголовок циклу
{ //тіло циклу
    if (i < 10)
    {
        сума += число;
        cout << i + 1 << ".\tСума = " << сума << endl;
        i++;
    }
    else break; // Переривання роботи циклу
}
```

Результати роботи всіх трьох прикладів будуть повністю ідентичні результату вихідного прикладу циклу `for`, в якому містились всі частини заголовку циклу.

Потрібно звернути увагу на те, що за відсутності у заголовку циклу умови виходу з циклу, цю умову потрібно прописати в тілі циклу та передбачити завершення циклу за допомогою *оператору break*. В іншому випадку цикл виконуватиметься нескінченно.

Приклад програмної реалізації

Приклад роботи з умовним оператором *if* та оператором ітераційного циклу *while* (Частина 1).

```
//Обчислення наближеного значення функції sin(pi/x)
//

#include "stdafx.h"
#include <iostream>
#include <windows.h> //Необхідно для виведення в консолі української мови
#include <math.h> //Бібліотека математичних функцій
using std::cout;
using std::cin;
using std::endl;

int main() //Розрахунок суми нескінченного ряду
{
    SetConsoleCP(1251); //Необхідно для виведення в консолі
                        //української мови
    SetConsoleOutputCP(1251); //Необхідно для виведення в консолі
                              //української мови
    double x, s = 0, s1=1, sk, p=2, fact=1;
    const double eps = 0.0001, pi=3.14;
    const int a = 180;
    int n = 0;
    cout << "Введіть значення X" << endl;
    cin >> x;
    //double s = 1 / x;
    if (abs(x)<=a && abs(x)>0) //розрхуємо значення суми ряду
                            //для x з певного діапазону  $0 < |x| \leq a$ 
    {
        sk = sin(pi/x); //значення контрольної суми
        while (abs(s - s1)>eps) //перевірка точності обчислень
                                //суми ряду
        {
            s1 = s;
            fact *= (2 * n + 1); //розрахунок факторіалу
            s += pow(-1, n)*(pow(pi, (2*n+1)))/(pow(x, (2 * n
                + 1))*fact);
            cout << "Ітерація " << n+1 << " поточне значення суми
                нескінченного ряду S=" << s << endl;
            n++;
        }
    }
    else
    {
        cout << "Значення має бути в межах  $0 < |x| \leq$ " << a << endl;
    }
}
```



```
}  
  
cout << "Контрольна сума sin(pi/x)= " << sk << endl;  
return 0;  
}
```

C++

Приклад роботи з умовним оператором *if* та оператором арифметичного циклу *for*, використання генератора випадкових чисел (Частина 2).

```
//Обчислення суми чисел у заданому діапазоні, підрахунок від'ємних
// чисел серед випадково згенерованих

#include "stdafx.h"
#include <iostream>
#include <random> //Бібліотека генерування випадкових чисел
#include <ctime> //Бібліотека, яка містить функцію time()
#include <windows.h> //Бібліотека, яка необхідна для виведення в консолі
української мови
using std::cout;
using std::cin;
using std::endl;

int main()
{
    SetConsoleCP(1251); //Необхідно для виведення в консолі
                        //української мови
    SetConsoleOutputCP(1251); //Необхідно для виведення в консолі
                              //української мови
    cout << "Введіть значення x" << endl;
    int x,y,lower,upper,sum=0;
    cin >> x;
    cout << "Введіть значення y" << endl;
    cin >> y;
    if (x>y)
    {
        upper = x;
        lower = y;
        cout << "x > y, x = " << x << endl;
    }
    else
    {
        upper = y;
        lower = x;
        cout << "x <= y, y = " << y << endl;
    }
    for (int i = lower; i <= upper; i++) //цикл рахує суму чисел від
                                        //меншого до більшого
    {
        sum += i;
        cout << "sum = " << sum << endl;
    }
    //Створює випадкове беззнакове ціле число
    std::default_random_engine e(time(0));
    //Повертає однорідно розподілене значення в заданому діапазоні
```



```
std::uniform_int_distribution<int> u(-lower, upper);
int n,z,neg=0;
cout << "Введіть кількість чисел для пошуку серед них
        від'ємних значень" << endl;
cin >> n;
for (int i = 0; i < n; i++) //цикл рахує кількість від'ємних чисел
{
    z=u(e); //випадкове ціле число з заданого діапазону
    cout << z << " ";
    if (z<0)
    {
        neg++;
    }
}
cout << endl;
cout << "Кількість від'ємних значень у наведеному вище рядку = "
    << neg << endl;
return 0;
}
```

C++

Завдання до комп'ютерного практикуму №3

Частина 1.

1. Розрахувати в'язкість аміаку в інтервалі температур від 12 до 18°C через 1°C за формулою Сатерленда

$$N = N_0 \frac{273,15 + C}{T \cdot C} \cdot (T/273,15)^n$$

При умові, що множник n визначається так:

$$n = A - B \cdot (t - 273,15) \cdot D \cdot 10^{-7} \cdot (t - 273,15)^2$$

де $A = 1,06$; $B = 1,04$; $D = 0$; $C = 503$; $N = 91,6 \cdot 10^{-7} \text{ Па} \cdot \text{с}$.

2. Розрахувати ізобарну теплоємність CO_2 в інтервалі температур від 30 до 150°C через 10°C за формулою:

$$C_p = E + F \cdot (T/100) + G \cdot (T/100)^2 + H \cdot (T/100)^3 + N \cdot (T/100)^4$$

де $E = 0,81513$; $F = 9,8454 \cdot 10^{-2}$; $G = -9,4747 \cdot 10^{-3}$; $H = 36,006 \cdot 10^{-5}$; $N = -0,0567$.

3. Використавши формулу завдання 2 при $E = 2,0183$; $F = 158,072 \cdot 10^{-2}$; $G = 273,61 \cdot 10^{-3}$; $H = -9380,9 \cdot 10^{-5}$; $N = -1,9986$, розрахувати ізобарну теплоємність аміаку в інтервалі температур від 80 до 150°C через 10°C.

4. За умови завдання 2 розрахувати ізохорну теплоємність CO_2 . Формула зв'язку має вигляд: $C_v = C_p - R/M$ (C – молекулярна маса газу; R – універсальна газова стала).

5. За умови завдання 2 розрахувати ізохорну теплоємність аміаку. Формула зв'язку має вигляд: $C_v = C_p - R/M$ (C – молекулярна маса газу; R – універсальна газова стала).

6. Розрахувати в'язкість води в інтервалі температур від 16 до 27°C через 1°C за формулою:

$$N_o = A(B + t)^n,$$

де $A = 0,59849$, $B = 43,252$, $n = -1,5423$.

7. Розрахувати теплопровідність аміаку в інтервалі температур від 16 до 27°C через 1°C за формулою:

$$L = A \frac{(T_K + c)}{(T + C)} \cdot \frac{T^{\frac{3}{2}}}{T_K^{\frac{3}{2}} M^{\frac{5}{6}}},$$

де $A = 4,3643 \cdot 10^{-3}$; $C = 503$; $T_K = 239,73$ К; M – молекулярна маса газу.

8. Розрахувати густину повітря в інтервалі температур від 45 до 54°C за формулою:

$$R = \frac{1,293 \cdot P}{(1 + 0,00367 \cdot t) \cdot 760}, \text{ кг/м}^3$$

де P – тиск, який дорівнює 105,2 кПа.

9. Розрахувати швидкість осадження кулеподібної частинки радіусом r ($r = 0,2 \cdot 10^{-3}$ м, $r = 0,3 \cdot 10^{-3}$ м ... $r = 0,7 \cdot 10^{-3}$ м) у газі. Використати формулу:

$$W_{oc} = \frac{d^2 \cdot R \cdot g}{18 \cdot M_c},$$

де $g = 9,81$ м/с²; $R = 1684$ кг/м³; $M_c = 5,2 \cdot 10^{-4}$ Н * с/м².

10. Розрахувати час процесу фільтрування суспензії об'ємом V ($V = 50 \cdot 10^{-6}$, $100 \cdot 10^{-6}$, $150 \cdot 10^{-6}$, $200 \cdot 10^{-6}$, $250 \cdot 10^{-6}$ м³) за формулою:

$$t = \frac{M \cdot r_0 \cdot x_0}{2p} \cdot \frac{V^2}{S^2} + \frac{M \cdot R}{P} \cdot \frac{V}{S},$$

де $M = 5,17 \cdot 10^{-9}$ Н * с/м³; $r = 4,15 \cdot 10^{12}$ м⁻²; $R = 2,3 \cdot 10^{10}$ м⁻¹; $x_0 = 0,178$ м³/м³; $P = 0,6 \cdot 10^5$ Н/м²; $S = 1$ м.

11. Розрахувати витрату потужності на перемішування для пропелерної мішалки діаметром $d = [2,6; 2,8; 3,0; 3,2]$ м. Використати формулу

$$N = K_N \cdot n^3 \cdot R_C \cdot d_m^5,$$

де $K_N = 0,32$; $n = 3,96$ об/с; $R_C = 1200$ кг/м³.

12. Розрахувати коефіцієнт теплопровідності L нітробензолу в інтервалі температур від 40 до 110°C через 10°C. Використати формулу

$$L_t = L_{30} \cdot [1 - E \cdot (t - 30)],$$

де $L_{30} = A_1 \cdot c \cdot R \cdot \sqrt{\frac{R}{M}}$, $A_1 = 4,22 \cdot 10^{-2}$; $c = 1,38 \cdot 10^3$ Дж/(кг*град); $R = 1200$ кг/м³; $M = 123$; $E = 1 \cdot 10^{-3}$ град⁻¹.

13. Розрахувати значення густини та в'язкості сірчаної кислоти в інтервалі температур від 10 до 80°C через 10°C. Використати формулу

$$R = 1894,8 - 0,909 \cdot t; \quad M = 1,406 \cdot 10^{-3} + 5,0087 \cdot 10^{-1}/t$$

14. Розрахувати опір шару осадка при фільтруванні суспензії з діаметром частинок $d_{cp} = (0,1; 0,15; 0,20; 0,25) \cdot 10^{-3}$ м за формулою:

$$r_{011} = A \cdot d_{cp}^B,$$

де $A = 9,8373 \cdot 10$; $B = -0,3224$.

15. Для десяти значень x перевірити, чи є рівняння

$$\sin^6 x + \cos^6 x = 1 - \frac{3}{4} \cdot \sin^2 2x$$

тотожністю.

16. Виконати завдання 16 для формули:

$$\tan x + \tan 2x - \tan 3x = -\tan x \cdot \tan 2x \cdot \tan 3x.$$

17. Виконати завдання 16 для формули:

$$\sin 2nA + \sin 2nB + \sin 2nG = (-1)^{n+1} \cdot 4 \cdot \sin nA \cdot \sin nB \cdot \sin nG,$$

де $n = 1, 2, 3, 4...$

18. Виконати 5-7 перевірок рівняння

$$\sin A + \sin B + \sin G = 4 \cdot \sin \frac{A+B}{2} \cdot \sin \frac{B+G}{2} \cdot \sin \frac{A+G}{2}$$

на тотожність, якщо A, B і G – кути трикутника.

19. Розрахувати залежність продуктивності вакуумфільтру W (кг/(час*м³)) від величини вакууму P (Н/м²), який визначається від 50 до 100 мм рт. ст. з кроком $P = 25$ мм рт. ст. для трьох розмірів $d = 0,25; 0,30; 0,35$ мм, якщо:

$$W = 7056 \cdot P^{0,5} \cdot d^2$$

20. Розрахувати тиск пару хлористого метилену P (та $\ln P$) в інтервалі температур 20...40°C через 5°C, якщо

$$\ln P = -\frac{A}{T} + B \cdot \ln T + C,$$

де $A = 1995,6$; $B = -3,4426$; $T = t + 273$; $C = 8,321; 8,621; 8,921$.

21. Розрахувати швидкість R хімічної реакції $A \rightarrow B$ у діапазоні температур $t = 300...400$ К із кроком 20 К, якщо

$$R = K_0 \cdot \exp\left(-\frac{E}{R \cdot T}\right) \cdot C_0,$$

де $K_0 = 2,05 \cdot \text{с}^{-1}$; $R = 1,98725 \text{ кал/моль} \cdot \text{град}$; $E = 20100 \text{ кал/моль}$; $C_0 = (2; 4; 6) \text{ моль/м}^3$.

22. Розрахувати, який об'єм займе 10 г азоту при тиску $P = 1,5; 3,0; 4,5 \text{ атм}$ і зміні температури від 200 до 400 K із кроком 10 K.

(Використати формулу 1; $R = 0,082057 \text{ л} \cdot \text{атм/моль} \cdot \text{град}$)

23. Розрахувати значення константи швидкості хімічної реакції

$$K = A \cdot \exp\left(\frac{-5,29 \cdot 10^4}{R \cdot T}\right)$$

В інтервалі температур 600 ... 800 через 50°C, якщо $A = 7,5 \cdot 10^{-5}; 8,0 \cdot 10^{-5}; 8,5 \cdot 10^{-5}$.

24. Розрахувати значення енергії Гіббса

$$G = -R \cdot T \cdot \ln K$$

хімічної реакції в інтервалі температур від 400 до 500 K із кроком 20 K, якщо $R = 1,3143 \text{ Дж/моль} \cdot \text{К}$, $K = 9,137; 9,237; 9,337$.

25. Розрахувати об'єм сегменту кулі

$$V = \frac{\pi \cdot h}{6} \cdot (h^2 + 3 \cdot r^2),$$

якщо h змінюється від 0,1 до 0,2 м із кроком 0,05 м, для $r = 0,5; 1,0; 1,5 \text{ м}$.

26. Обчислити та запам'ятати значення:

$$y_{ij} = A_i^{2,2} + 1,5 \cdot A_i \cdot B_i \cdot C_j - B_i^{1,5} \cdot \sqrt{A_i + 2B_i}; \quad A_i = 1,8 \cdot \cos^3 x_i; \quad B_i = e^{\frac{2}{x_i}} + 2,6.$$

а потім знайти суму всіх отриманих значень функції y_{ij} .

Прийняти $x_i = 2,2; 1,9; 1,7; 1,4; 1,2$; $C_j = 2,31; 2,86; 3,57$.

27. Обчислити та запам'ятати значення $z_i = F(x_i, y_i)$ в n різних точках (x_i, y_i) , а потім знайти суму всіх отриманих значень функції. Прийняти $n > 10$; $x_{i+1} = x_i + \delta x$ ($x_0 = 2,8$; $\delta x = 2,1$); $y_{i+1} = y_i + \delta y$ ($y_0 = 1,7$; $\delta y = 0,3$).

$$F(x_i, y_i) = [\ln(x_i^2) - \ln(y_i^2)] \frac{\sqrt[3]{x_i - y_i}}{\text{ctg}(x_i)}.$$

28. Обчислити значення:

$$y_i = a_i^{2b_i} + b_i^{3a_i}; \quad a_i = e^{2,5/x_i} + 3x_i \cdot C; \quad b_i = \sin^2 x_i + x_i^{0,5}.$$

Прийняти $x_i = 0,3; 0,7; 0,9; 1,1; 1,2; 1,4; 1,7; 1,8; 1,9; 2,2; 2,5; 2,7$; $C = 2,15$.

29. Розрахувати значення тригонометричних функцій $\sin(A)$, $\cos(A)$ і $\text{ctg}(A)$ при зміні A від 30° до 150° з кроком 15° .

30. Знайти перші десять та обчислити суму перших трьох, п'яти та дев'яти членів ряду, у якому

$$A_n = \frac{2n! - (-1)^n \cdot 2^{n-1}}{2^{n+1} + n!}$$



Частина 2.

1. $S \approx \frac{1}{x} + 1 + \frac{x}{2!} + \frac{x^2}{3!} + K = \frac{e^x}{x} \quad |x| < 1$
2. $S \approx 1 + \frac{1}{4}x + \frac{1}{4} \cdot \frac{5}{8} \cdot x^2 + \frac{1}{4} \cdot \frac{5}{8} \cdot \frac{9}{12} x^3 + \dots = (1-x)^{-1/4} \quad |x| < 1$
3. $S \approx 1 + \frac{1}{2} \cdot x + \frac{1 \cdot 3}{2 \cdot 4} \cdot x^2 + \frac{1 \cdot 3 \cdot 5}{2 \cdot 4 \cdot 6} \cdot x^3 + K = (1-x)^{-\frac{1}{2}} |x| < 1$
4. $S \approx -x - \frac{x^2}{2} - \frac{x^3}{3} - \frac{x^4}{4} - \dots = \ln(1-x) \quad -1 \leq x < 1$
5. $S \approx x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \dots = \arctg(x) \quad |x| \leq 1$
6. $S \approx 1 - x + x^2 - x^3 + K = \frac{1}{1+x} \quad |x| < 1$
7. $S \approx 1 + \frac{1}{3}x + \frac{1 \cdot 4}{3 \cdot 6} x^2 + \frac{1 \cdot 4 \cdot 7}{3 \cdot 6 \cdot 9} x^3 + \dots = (1-x)^{-1/3} \quad |x| < 1$
8. $S \approx 2 \left(x + \frac{x^3}{3} + \frac{x^5}{5} + \frac{x^7}{7} + K \right) = \ln \frac{1+x}{1-x} \quad |x| < 1$
9. $S \approx x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \dots = \ln(1+x) \quad -1 < x \leq 1$
10. $S \approx 1 + x + x^2 + x^3 + \dots = \frac{1}{1-x} \quad |x| < 1$
11. $S \approx 1 - x^2 + x^4 - x^6 + x^8 - K = \frac{1}{1+x^2} \quad |x| < 1$
12. $S \approx 1 + \frac{1}{x} + \frac{1}{2!x^2} + \frac{1}{3!x^3} + \dots = e^{1/x} \quad |x| \leq 1$
13. $S \approx 1 - 2x + 3x^2 - 4x^3 + 5x^4 - K = \frac{1}{(1+x)^2} \quad |x| < 1$
14. $S \approx 1 - \frac{1}{4} \cdot x + \frac{1}{4} \cdot \frac{5}{8} \cdot x^2 - \frac{1}{4} \cdot \frac{5}{8} \cdot \frac{9}{12} \cdot x^3 + K = (1+x)^{-1/4} \quad |x| < 1$
15. $S \approx x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots = \sin(x) \quad |x| \leq \infty$
16. $S \approx 1 - \frac{5}{2}x + \frac{5 \cdot 7}{2 \cdot 4} x^2 - \frac{5 \cdot 7 \cdot 9}{2 \cdot 4 \cdot 6} x^3 + \dots = (1+x)^{-5/2} \quad |x| < 1$
17. $S \approx 1 - x + \frac{x^2}{2!} - \frac{x^3}{3!} + \dots = e^{-x} \quad |x| \leq \infty$
18. $S \approx 1 - \frac{1}{3}x + \frac{1}{3} \cdot \frac{4}{6} x^2 - \frac{1}{3} \cdot \frac{4}{6} \cdot \frac{7}{9} x^3 + \dots = (1+x)^{-1/3} \quad |x| < 1$
19. $S \approx 1 - \frac{1}{2} \cdot x + \frac{1 \cdot 3}{2 \cdot 4} \cdot x^2 - \frac{1 \cdot 3 \cdot 5}{2 \cdot 4 \cdot 6} \cdot x^3 + \dots = (1+x)^{-\frac{1}{2}} \quad |x| < 1$

$$20. S \approx 1 - \frac{3}{2} \cdot x + \frac{3 \cdot 5}{2 \cdot 4} \cdot x^2 - \frac{3 \cdot 5 \cdot 7}{2 \cdot 4 \cdot 6} \cdot x^3 + K = (1+x)^{-3/2} \quad |x| < 1$$

$$21. S \approx 1 - \frac{2 \cdot 3}{2} x + \frac{3 \cdot 4}{2} x^2 - \frac{4 \cdot 5}{2} x^3 + \dots = \frac{1}{(1+x)^3} \quad |x| < 1$$

$$22. S \approx 1 + \frac{3}{2} \cdot x + \frac{3 \cdot 5}{2 \cdot 4} \cdot x^2 + \frac{3 \cdot 5 \cdot 7}{2 \cdot 4 \cdot 6} \cdot x^3 + K = (1-x)^{-3/2} \quad |x| < 1$$

$$23. S \approx \frac{1}{x} - \left(\frac{x}{3} + \frac{x^3}{45} + \frac{2x^5}{945} + \frac{x^7}{4725} + \dots \right) = ctg(x) \quad (0 < |x| < \pi)$$

$$24. S \approx -\frac{x^2}{2} - \frac{x^4}{12} - \frac{x^6}{45} - \frac{17x^8}{2520} - \dots = \ln |\cos(x)| \quad (|x| < \frac{\pi}{2})$$

$$25. S \approx x + \frac{x^3}{3!} + \frac{x^5}{5!} + \frac{x^7}{7!} + K = sh(x) \quad |x| \leq \infty \quad \left[sh(x) = \frac{e^x - e^{-x}}{2} \right]$$

$$26. S \approx 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \dots = \cos(x) \quad |x| \leq \infty$$

$$27. S \approx 1 + \frac{x^2}{2!} + \frac{x^4}{4!} + \frac{x^6}{6!} + K = ch(x) \quad |x| \leq \infty \quad \left[ch(x) = \frac{e^x + e^{-x}}{2} \right]$$

$$28. \text{Розрахувати } k \text{ членів ряду } S \approx x + \frac{x+1}{2x} + \frac{x+2}{3x^2} + K \quad |x| \leq \infty$$

$$29. \text{Розрахувати } k \text{ членів ряду } S \approx 1 + \frac{1}{x} + \frac{1-x}{2!x^2} + \frac{1-2x}{3!x^3} + \frac{1-3x}{4!x^4} \dots \quad |x| \leq 1$$

$$30. \text{Розрахувати } k \text{ членів ряду } S \approx 1 - \frac{x^2}{x-2!} + \frac{x^4}{x-3!} - \frac{x^6}{x-4!} + \frac{x^8}{x-5!} - K \quad |x| < 1$$

Контрольні питання

- 1) Опишіть структуру та принцип роботи оператора while?
- 2) Що таке блок? Наведіть приклад.
- 3) Опишіть структуру та принцип роботи оператора for?
- 4) З чого складається заголовок оператора for?
- 5) У чому відмінність між операторами циклу for та while?
- 6) Для чого використовуються об'єкти класу default_random_engine?
- 7) Як згенерувати випадкові цілі числа в заданому діапазоні?
- 8) Як згенерувати випадкові дійсні числа в заданому діапазоні?
- 9) Як забезпечити генерування різних послідовностей випадкових чисел при кожному запуску програми?



Комп'ютерний практикум №4

Змінні та базові типи

Мета: ознайомитись із символьними та чисельними обчисленнями засобами C++; вивчити існуючі типи даних, структуровані типи даних (*вказівники, посилання, створення власної структури даних*); навчитись створювати *файли заголовку*; відпрацювати засоби візуалізації розрахунків у C++. Навчитись використовувати кирилицю в консолі: коректне розпізнавання введених символів кирилицею; коректне виведення в консолі символів кирилицею.

Завдання: створити консольний додаток для реалізації обчислень поставленої задачі згідно двох частин отриманого варіанту завдання з використанням створеної структури даних та покажчиків і посилань.

Загальні вимоги.

- 1) Створити консольний додаток з ім'ям виду *PrizvischeKP4*.
- 2) Програмний код модуля має бути чітко структурований.
- 3) Імена об'єктів мають нести сенсові навантаження.
- 4) Програмний код має супроводжуватись коментарями в тексті програми.

Вимоги до виконання.

Частина 1.

- 1) Створити власну *структуру даних* з ім'ям виду *PrizvischeKP4_z1*, яка включатиме в якості полів даних всі необхідні вхідні та вихідні параметри задачі з обов'язковою наявністю *цілочисельних, дійсних та рядкових типів*, а також *константних змінних*.
- 2) Визначити власну *структуру даних* у *файлі заголовку* з ім'ям виду *PrizvischeKP_z1.h*.
- 3) Створити відповідні типи *посилань* та *посилань на константу* для кожного поля створеної *структури даних*.
- 4) Запрограмувати вирішення поставленої задачі з використанням лише *посилань* на поля *структури*.
- 5) Введення вхідних даних організувати в режимі діалогу з користувачем.
- 6) Завдання для обчислень, вхідні дані та результати розрахунків вивести у зовнішній текстовий файл з ім'ям виду *ПрізвищеКП4_z1_OUT*.

- 7) При виведенні вхідних даних та результатів розрахунків використовувати безпосередньо звернення до полів даних створеної *структури* замість *посилань* на них.
- 8) Виведення даних у вихідних файл організувати в режимі додавання в кінець файлу.

Частина 2.

- 1) Створити власну *структуру даних* з ім'ям виду *PrizvischeKP4_z2*, яка включатиме в якості полів даних всі необхідні вхідні та вихідні параметри задачі з обов'язковою наявністю *цілочисельних, дійсних та рядкових типів*, а також *константних змінних*.
- 2) Визначити власну *структуру даних* у файлі заголовку з ім'ям виду *PrizvischeKP4_z2.h*.
- 3) Створити відповідні типи *показчиків* та *показчиків на константу* для кожного поля створеної *структури даних*.
- 4) Запрограмувати вирішення поставленої задачі з використанням лише *показчиків* на поля *структури*.
- 5) Введення вхідних даних організувати в режимі діалогу з користувачем.
- 6) Завдання для обчислень, вхідні дані та результати розрахунків вивести у зовнішній текстовий файл з ім'ям виду *ПрізвищеКП4_z2_OUT*.
- 7) При виведенні вхідних даних та результатів розрахунків використовувати безпосередньо звернення до полів даних створеної *структури* замість *показчиків* на них.
- 8) Виведення даних у вихідний файл організувати в режимі заміни вмісту файлу.

Теоретичні відомості

Тема 4.1. Складені типи

Складений тип (compound type) – це тип, який визначений у термінах іншого типу. У мові C++ є декілька складених типів, два з яких це посилання та показники.

4.1.1 Посилання

Посилання (reference) є альтернативним іменем об'єкту. Тип посилання «посилається на» інший тип. У визначенні типу посилання використовується

оператор оголошення у формі `&d`, де `d` – ім'я, що об'являється. Символ `&` слід повторювати для кожного імені покажчика.

```
int &a, &b, c; //c не є посиланням
int i;
int &j = i;
```

Зазвичай при ініціалізації змінної значення ініціалізатора копіюється в створюваний об'єкт. При визначенні посилання замість копіювання значення ініціалізатора відбувається зв'язування (bind) посилання з його ініціалізатором. Після ініціалізації посилання залишається зв'язаним із вихідним об'єктом. Неможливо змінити прив'язку посилання і зв'язати його з іншим об'єктом, тому посилання обов'язково слід ініціалізувати.

Посилання – це не об'єкт, а тільки *інше ім'я (псевдонім) вже існуючого об'єкту*. Загальні правила щодо використання посилань:

- після того, як посилання визначене, всі операції з ним фактично здійснюються з об'єктом, із яким пов'язане посилання;
- при присвоєнні посилання присвоюється об'єкт, із яким воно пов'язане;
- при доступі до значення посилання фактично відбувається звернення до значення об'єкту, з яким пов'язане посилання;
- коли посилання використовується як ініціалізатор, у дійсності для цього використовується об'єкт, із яким пов'язане посилання;
- оскільки посилання не об'єкти, то неможливо визначити посилання на посилання;
- типи посилання та об'єкту, на який воно посилається мають співпадати точно;
- посилання може бути пов'язане лише з об'єктом, але не з літералом чи результатом більш загального виразу.

```
int i;
float &k = i; //Помилка! Неспівпадіння типів посилання та об'єкту
int &j = 10; //Помилка! Посилання на літерал
```

4.1.2 Покажчик

Покажчик (pointer) – це складений тип, змінна якого вказує на об'єкт іншого типу. Подібно посиланням, покажчики використовують для опосередкованого доступу до інших об'єктів. Загальні правила щодо використання покажчиків:

- на відміну від посилання, покажчик – це справжній об'єкт;
- покажчики можуть бути присвоєні та скопійовані;

- один покажчик за час свого існування може вказувати на кілька різних об'єктів;
- на відміну від посилання, покажчик можна не ініціалізувати у момент визначення;
- типи покажчика та об'єкту, яким він ініціалізується мають співпадати;
- оскільки посилання не об'єкти, в них немає адрес, а відповідно, неможливо визначити покажчик на посилання.

Тип покажчика визначається оператором у формі `*d`, де `d` – ім'я покажчика, що створюється. Символ `*` слід повторювати для кожної змінної покажчика.

```
int *p1, *p2, c; //c не є покажчиком
```

Покажчик містить адресу іншого об'єкту. Для отримання адреси об'єкту використовують *оператор звернення до адреси* (address-of operator), або оператор `&`.

```
int i=77;  
int *p=&i; //p містить адресу змінної i, p - покажчик на змінну i
```

У прикладі `p` визначене як покажчик на тип `int` та ініціалізоване адресою об'єкту `i` типу `int`.

Коли покажчик вказує на об'єкт, для доступу до цього об'єкту можна використати *оператор звернення до значення* (dereference operator), або оператор `*`.

```
int i=28;  
int *p=&i; //p містить адресу змінної i, p - покажчик на змінну i  
cout << *p; /* повертає об'єкт, на який вказує p, виводиться 28
```

Звернення до значення покажчика (`*`) повертає об'єкт, на який вказує покажчик.

Деякі символи в C++ можуть використовуватись в різному сенсі. Так символи `&` і `*` використовуються і як оператор у виразах, і як частина оголошення. Контекст, в якому використовується символ, визначає його значення.

```
int i = 40;  
//& стоїть після типу в частині оголошення i оголошує посилання  
int &посил = i;  
/* стоїть після типу в частині оголошення i оголошує покажчик  
int *покажч;  
//& використовується у виразі як оператор звернення до адреси  
покажч = &i;  
/* використовується у виразі як оператор звернення до значення  
*покажч = i;
```



```
//& є частиною оголошення типу, * - оператор звернення до значення
int &посил2 = *покажч;
//Виводимо значення об'єкту, на який вказує посилання
std::cout << посил << std::endl; // 40
//Виводимо значення об'єкту, на який вказує покажчик
std::cout << *покажч << std::endl; // 40
//Виводимо адресу об'єкту в пам'яті, на який вказує покажчик
std::cout << покажч << std::endl; // 00E5F934
```

В оголошеннях символи & і * використовуються для формування складених типів. У виразах ці ж символи використовуються для позначення оператора. Оскільки ті ж самі символи використовуються в різному контексті з різним сенсом, їх слід сприймати, як абсолютно різні символи.

Нульовий покажчик (null pointer) не вказує на жоден об'єкт. Код може перевіряти, чи не є покажчик нульовим, перед тим як його використовувати. Існує декілька способів отримати нульовий покажчик.

```
int *покажч1 = nullptr; // еквівалентно int *покажч1 = 0;
int *покажч2 = 0;       // безпосередньо ініціалізує покажч2
                        //літеральною константою 0
int *покажч3 = NULL;    // еквівалентно int *покажч3 = 0;
```

Краще за все ініціалізувати нульовий покажчик літералом nullptr. В якості альтернативи можна ініціалізувати покажчик літералом 0. Деякі програмісти можуть використовувати *змінну препроцесора* (preprocessor variable) NULL, яку заголовок cstdlib визначає як 0, але таке визначення нульового покажчика є небажаним.

І покажчика, і посилання надають опосередкований доступ до інших об'єктів. Однак є важливі відмінності у способі, яким вони це роблять. Найважливіше то, що посилання не є об'єктом. Після визначення посилання немає ніякого способу примусити його посилатись на інший об'єкт. При використанні посилання завжди використовується об'єкт, з яким воно було пов'язане спочатку.

Між покажчиком і адресою, яку він містить, немає такого зв'язку. Подібно будь-якій змінній, при присвоєнні покажчика для нього встановлюється нове значення. Присвоєння примушує покажчик вказувати на інший об'єкт.

4.1.3 Специфікатор const

Іноколи потрібно визначити змінну, значення якої не може бути змінено. Для визначення змінних, значення яких не може мінятися, використовується *специфікатор const* (qualifier const):

```
const int bufSize = 512; // розмір буферу введення
```


Це визначить змінну `bufSize` як константу. Будь-яка спроба присвоїти їй значення буде помилкою:

```
bufSize = 1024; // Помилка! Спроба запису у константний об'єкт
```

Оскільки неможливо змінити константний об'єкт, він обов'язково має ініціалізований при створенні. Ініціалізатор, як завжди, може бути виразом будь-якої складності:

```
//Ініціалізація значенням функції під час виконання
const int i = get_size();
//Ініціалізація числовим літералом під час компіляції
const int j = 38;
const int k; // Помилка! k - неініціалізована константа
```

Тема 4.2. Робота з типами

З ускладненням програм, типи, які в них використовуються, теж стають складнішими. Ускладнення у використанні типів виникають з двох причин. Імена деяких типів складно писати по пам'яті. Написання деяких їх форм складне і може викликати помилки. Крім того, формат запису складного типу може приховати його ціль чи призначення. Інше джерело ускладнень пов'язане з тим, що інколи складно напевно визначити необхідний тип. Це може вимагати перегляду контексту програми.

4.2.1 Псевдоніми типів

Псевдонім типу (type alias) – це ім'я, що є синонімом імені іншого типу. Псевдоніми типів дозволяють спростити складні визначення типів, що полегшує їх використання. Псевдоніми типів дозволяють також підкреслювати мету використання типу. Визначити псевдонім типу можна одним з двох способів. Традиційно для цього використовують ключове слово `typedef`:

```
typedef size_t лічильник;
лічильник i; //Змінна i має тип size_t
typedef unsigned char символ;
символ ch; //Змінна ch має тип беззнакового символу unsigned char
typedef long double число;
число x; // Змінна x має тип числа з плаваючою комою
//підвищеної точності long double
число &y = x; // Змінна y є посиланням типу числа з плаваючою комою
//підвищеної точності long double
символ a, *b; //Змінна a має тип unsigned char, а змінна b
//є покажчиком на тип unsigned char
x = 67.89775;
```



```
printf("x = %f\t y = %f\n", x, y);
```

Результат роботи прикладу:

```
x = 67.897750    y = 67.897750
```

В даному прикладі застосована функція `printf`, яка друкує форматоване виведення у стандартний потік виведення.

Ключове слово `typedef` може бути частиною базового типу в оголошенні. Оголошення, які містять ключове слово `typedef`, визначають псевдоніми типу, а не змінні. Як і інші оголошення, `typedef` може містити модифікатори типу, які визначають складені типи, що містять базовий тип.

В стандарті C++11 введено інший спосіб визначення псевдоніму типу за допомогою оголошення псевдоніму (*alias declaration*) *using* і знаку `=`.

```
char a = 'v';
using чсл = unsigned int;
using смвл = char&;
чсл i; //Змінна i має тип unsigned int
смвл c = a; //Змінна c є посиланням на тип char
i = c;
using дійсн = long double;
дійсн z; //Змінна z має тип long double
printf("a = %c\t c = %c\t i = %d\n", a, c, i);
```

Результатом роботи прикладу буде наступний рядок виведення:

```
a = v    c = v    i = 118
```

Псевдонім типу – це ім'я типу і воно може міститись кругом, де присутнє ім'я типу.

C++11 — чинна версія стандарту мови C++, прийнята у серпні 2011 комітетом ISO зі стандартизації мови замість ISO/IEC 14882:2003 (C++03). Новий стандарт включає доповнення в ядрі мови та розширення STL, включаючи велику частину TR1 — окрім, можливо, бібліотеки спеціальних математичних функцій.

C++ Technical Report 1 (TR1) є загальною назвою для стандарту ISO/IEC TR 19768, бібліотеки розширень C++ - це документ з пропозицією доповнень до стандарт бібліотеки C++. Додатки включають регулярні вирази, розумні покажчики, хеш-таблиці, і генератори випадкових чисел. TR1 не стандарт, а швидше проект документа. Однак, більшість його пропозицій, стали частиною такого офіційного стандарту, як C++11.

4.2.2 Специфікатором типу auto

Коли складно при оголошенні об'єкту визначити тип його можливих значень, можна скористатися *специфікатором типу auto*, який з'явився у стандарті C++11. На відміну від інших специфікаторів типів (`int`, `double`), які задають певний тип, специфікатор `auto` наказує компілятору вивести тип з ініціалізатора. У зв'язку з цим, у змінної що використовує специфікатор `auto`, має бути ініціалізатор.

```
//item ініціалізується результатом суми val1 + val2
auto item = val1 + val2;
//специфікатор auto може визначати кілька змінних
auto i = 0, *p = &i;
```

4.2.3 Специфікатором типу decltype

Для визначення змінної, тип якої компілятор виводить із виразу, але не використовує цей вираз для ініціалізації змінної, застосовують *специфікатор типу decltype*. Специфікатор `decltype` з'явився у стандарті C++11 і повертає тип його операнду.

```
decltype(f()) sum = x;
```

Тут компілятор не викликає функцію `f()`, але він використовує тип, який повернув би такий виклик для змінної `sum`. Таким чином, компілятор призначає змінній `sum` той же тип, який був би повернутий при виклику функції `f()`.

Коли вираз, до якого застосовується специфікатор `decltype`, є змінною, він повертає тип цієї змінної, включаючи специфікатор `const` верхнього рівня та посилання.

```
const int змінна = 0, &посилання = змінна;
decltype(змінна) x = 0; // x має тип const int
decltype(посилання) y = x; // y має тип const int& і пов'язаний з x
decltype(посилання) z;    // Помилка! Змінна z є посиланням, тому вона
                          // обов'язково має бути ініціалізована
```

Оскільки `змінна &посилання` – є посиланням, `decltype(посилання)` має тип посилання. Як і будь-яке посилання, посилання `z` потрібно ініціалізувати.

Коли специфікатор `decltype` застосовується до виразу, який не є змінною, отриманий тип відповідає типу виразу.

```
int i = 28, &j = i;
decltype(j) x;    // Помилка! Змінна x є посиланням, тому вона обов'язково
                  //має бути ініціалізована
```



```
decltype(j+0) z;      // Результат виразу має тип int,  
                      //тому i змінна z має тип int
```

Відмінність між специфікаторами `auto` та `decltype` полягає в тому, що виведення типу специфікатором `decltype` залежить від форми виразу. Включення імені змінної в круглі дужки впливає на тип, який повертає специфікатор `decltype`. При застосуванні специфікатора `decltype` до змінної без круглих дужок отримуємо тип цієї змінної. Якщо заключити ім'я змінної у дужки, то компілятор буде розуміти операнд як вираз. Змінна – це вираз, який може бути лівим операндом присвоєння (лівосторонній оператор). В результаті специфікатор `decltype` для такого виразу поверне посилання.

```
int i = 6;  
decltype(i) z; // Змінна z має тип int  
decltype((i)) x; // Помилка! Змінна x є посиланням, тому вона обов'язково  
                //має бути ініціалізована  
decltype((i)) x=i; //Змінна x має тип &int та посилається на змінну i
```

Тема 4.3. Визначення власних структур даних

4.3.1 Ключове слово `struct`

У мові C++ можна створювати власні типи даних, створюючи клас. На найпростішому рівні *структура даних* (data structure) – це спосіб групування взаємопов'язаних даних та стратегії їх використання.

Визначення класу починається із *ключового слова* `struct`, яке супроводжується ім'ям класу та (можливо пустим) тілом класу. *Тіло класу* обмежується фігурними дужками і формує нову *область видимості*. Визначені у класі імена мають бути унікальними в межах класу, але поза класом вони можуть повторюватись.

```
struct MyStruct //Створення власної структури даних (класу)  
{  
    std::string назва; // ініціалізатор за замовчуванням  
    unsigned индекс = 1; // внутрішньокласовий ініціалізатор  
    float сума = 1.0; // внутрішньокласовий ініціалізатор  
    vector<int> вектор{ 1,2,3,4,5 }; //внутрішньокласовий ініціалізатор  
                                // із використанням фігурних дужок  
};
```

За фігурною дужкою, що закриває тіло класу, має бути крапка з комою. Крапка з комою необхідна, оскільки після тіла класу можливо визначити змінні.

У тілі класу визначені члени (member) класу. У нашому прикладі у класу є лише *змінні-члени* (data member). Змінні-члени класу визначають вміст об'єктів цього класу. Кожен об'єкт класу має власний екземпляр змінних-членів класу.

У змінних-членів класу можна визначити *внутрішньокласовий ініціалізатор* (in-class initializer). Він використовується для ініціалізації змінних-членів при створенні об'єктів. Члени без ініціалізаторів ініціалізуються за замовчуванням. Внутрішньокласові ініціалізатори мають бути укладені у фігурні дужки або йти за знаком =. Неможна визначити внутрішньокласовий ініціалізатор у круглих дужках.

```
unsigned індекс = 1; //внутрішньокласовий ініціалізатор
float сума = 1.0; //внутрішньокласовий ініціалізатор
vector<int> вектор{ 1,2,3,4,5 }; //внутрішньокласовий ініціалізатор
// із використанням фігурних дужок
```

Мова C++ використовує для визначення власних структур окрім ключового слова struct також ключове слово class, яке буде розглянуте пізніше.

Для того, щоб скористатися створеною структурою даних, необхідно визначити об'єкт типу створеної структури. *Визначення об'єкту* типу класу в найпростішому випадку може бути аналогічне визначенню об'єктів інших типів.

```
string слово; //визначення змінної вбудованого типу string
MyStruct моя_структура; //визначення змінної власного типу MyStruct
```

Після визначення змінної типу класу стає можливим звернення до членів даного класу. Звернення до членів класу використовується для отримання значень змінних-членів, зміни (присвоєння нового значення) значень змінних-членів, виконання певних операцій, які передбачені даним класом. Звернення до членів класу виконується за допомогою *оператору звернення до члену* (.).

```
int i = моя_структура.індекс; //значення i=1
моя_структура.сума = 5.5;
vector<int> vect;
vect = моя_структура.вектор; //vect={ 1,2,3,4,5 }
```

Класи зазвичай визначають у *файлах заголовку*. Хоча можливе визначення у найпростіших випадках безпосередньо перед функцією main. Повний текст попереднього прикладу наведений нижче.

```
#include "stdafx.h"
#include <string>
#include <vector>
using std::string;
using std::vector;
```



```
struct MyStruct //Створення власної структури даних (класу)
{
    string назва; // ініціалізатор за замовчуванням
    unsigned індекс = 1; // внутрішньокласовий ініціалізатор
    float сума = 1.0; // внутрішньокласовий ініціалізатор
    vector<int> вектор{ 1,2,3,4,5 }; //внутрішньокласовий ініціалізатор
                                   // із використанням фігурних дужок
};

int main()
{
    MyStruct моя_структура; //визначення змінної власного типу MyStruct
    int i = моя_структура.індекс; //значення i=1
    моя_структура.сума = 5.5;
    vector<int> vect;
    vect = моя_структура.вектор; //vect={ 1,2,3,4,5 }
    return 0;
}
```

4.3.2 Файли заголовку header

Файли заголовку (header) мають розширення *.h. Як правило, класи зберігаються в заголовках, ім'я яких співпадає з іменем класу. Наприклад, бібліотечний тип string визначений у заголовку string. За аналогією наш клас MyStruct має бути визначений у заголовку MyStruct.h.

Заголовки підключаються до програмного файлу за допомогою директиви препроцесора #include. Коли препроцесор зустрічає директиву #include, він замінює її вмістом файлу заголовку, який підключений за допомогою цієї директиви. З огляду на це, заголовки також не повинні містити оголошення using, щоб запобігти включенню в програму не передбачених бібліотечних імен.

4.3.3 Захист заголовку: директиви #define, #ifdef та #ifndef

Заголовки містять сутності, які можуть бути визначені у будь-якому файлі лише раз. Для запобігання кількаразового включення вмісту одного й того самого заголовку, програми C++ використовують препроцесор для захисту заголовку (header guard). Захист заголовку опирається на змінні препроцесору. Змінні препроцесору можуть знаходитись у двох станах: визначена чи не визначена. Директива #define отримує ім'я і визначає його як змінну препроцесору. Існують дві директиви, що дозволяють перевірити чи була визначена змінна препроцесора. Директива #ifdef істинна, якщо змінна була визначена, а директива #ifndef істинна, якщо змінна не була визначена. При

істинності перевірки виконується все, що розташоване після директиви `#ifdef` або `#ifndef` і до наступної директиви `#endif`.

Ці засоби можна використовувати для боротьби з багаторазовим включенням вмісту файлів заголовку наступним чином

```
#ifndef MY_STRUCT_H
#define MY_STRUCT_H
#include <string>
struct MyStruct //Створення власної структури даних (класу)
{
    std::string назва;
    unsigned індекс = 1;
    float сума = 0.0;
};
#endif
```

При першому включенні заголовку `MyStruct.h` директива `#ifndef` істинна, і препроцесор обробляє рядки після неї до директиви `#endif`. У результаті змінна `MY_STRUCT_H` буде визначена, а вміст заголовку `MyStruct.h` скопійовано до програми. Якщо надалі включити заголовок `MyStruct.h` у той же самий файл, то директива `#ifndef` виявиться брехнею і рядки між нею та директивою `#endif` будуть проігноровані.

Змінні препроцесора мають бути унікальними в усій програмі. Для цього в ім'я змінної препроцесора включається ім'я класу та ім'я змінної препроцесора задається лише великими літерами. Наприклад змінна препроцесора `MY_STRUCT_H` для включення визначення класу `MyStruct` із заголовку `MyStruct.h`.

Кожен заголовок повинен містити захист від повторного включення до програми його вмісту.

Приклад програмної реалізації

Приклад роботи зі структурами та функціями.

//Визначення знаку зодіаку людини за датою її народження

```
#include "stdafx.h"
#include <string>
#include <iostream>
#include "windows.h"
#include "BirthdayStruct.h"
#include "Zodiac.h"
using std::cin;
using std::cout;
using std::endl;
using std::string;

int _tmain(int argc, _TCHAR* argv[]) //Створення власної структури даних
//та функції користувача з підключенням їх
{
    //через файли заголовку, використання посилань
    SetConsoleCP(1251); //Необхідно для виведення в консолі
                        //української мови
    SetConsoleOutputCP(1251); //Необхідно для виведення в консолі
                        //української мови
    BirthdayStruct Birthday; //Створення змінною типу власної
                        //структури (класу) BirthdayStruct
    cout << "Введіть ім'я людини" << endl;
    cin >> Birthday.bName; //Введення даних в поля структури
    cout << "Введіть число дня народження" << endl;
    cin >> Birthday.bDay;
    cout << "Введіть номер місяця народження" << endl;
    cin >> Birthday.bMonth;
    cout << "Введіть рік народження" << endl;
    cin >> Birthday.bYear;
    string &name = Birthday.bName;
    unsigned &day = Birthday.bDay;
    unsigned &month = Birthday.bMonth;
    unsigned &year = Birthday.bYear;
    //Виклик функції fzodiac з передачею їй параметрів посилань
    cout << name << " за гороскопом " << fzodiac(month, day) << endl;
    return 0;
}
```


Приклад створення зовнішньої функції.

```
//zodiacFunc.cpp          Файл з кодом зовнішньої функції

#include "stdafx.h"
#include <string>
#include "Zodiac.h"

std::string fzodiac(const unsigned &fMonth, const unsigned &fDay)
//Визначення власної функції з параметрами посиланнями
//на константи типу unsigned
{
    std::string zodiac;
    switch (fMonth) //Використання оператора switch для визначення
    //знаку зодіаку в залежності від місяця та дня народження
    {
        case 1:
            fDay <= 20 ? zodiac = "Козеріг" : zodiac = "Водолій";
            break;
        case 2:
            fDay <= 18 ? zodiac = "Водолій" : zodiac = "Риби";
            break;
        case 3:
            fDay <= 20 ? zodiac = "Риби" : zodiac = "Овен";
            break;
        case 4:
            fDay <= 20 ? zodiac = "Овен" : zodiac = "Телець";
            break;
        case 5:
            fDay <= 21 ? zodiac = "Телець" : zodiac = "Близнюки";
            break;
        case 6:
            fDay <= 21 ? zodiac = "Близнюки" : zodiac = "Рак";
            break;
        case 7:
            fDay <= 22 ? zodiac = "Рак" : zodiac = "Лев";
            break;
        case 8:
            fDay <= 23 ? zodiac = "Лев" : zodiac = "Діва";
            break;
        case 9:
            fDay <= 23 ? zodiac = "Діва" : zodiac = "Терези";
            break;
        case 10:
```

```
        fDay <= 23 ? zodiac = "Терези" : zodiac = "Скорпіон";  
        break;  
    case 11:  
        fDay <= 22 ? zodiac = "Скорпіон" : zodiac = "Стрілець";  
        break;  
    case 12:  
        fDay <= 21 ? zodiac = "Стрілець" : zodiac = "Козеріг";  
        break;  
};  
return zodiac; //Значення, яке повертає функція  
              //у місце свого виклику  
}
```

C++

Приклад створення власної структури даних.

// BirthdayStruct.h Файл заголовку з оголошенням власної структури даних

```
#pragma once
```

```
#ifndef BIRTHDAYSTRUCT_H
```

```
#define BIRTHDAYSTRUCT_H
```

```
#include <string>
```

```
struct BirthdayStruct //Створення власної структури даних (класу)
```

```
{
```

```
    std::string bName;
```

```
    unsigned bDay;
```

```
    unsigned bMonth;
```

```
    unsigned bYear;
```

```
    float bAge;
```

```
    std::string bZodiac;
```

```
};
```

```
#endif
```

C++

Приклад оголошення зовнішньої функції у файлу заголовку.

//Zodiac.h Файл заголовку з оголошенням зовнішньої функції

```
#pragma once
#ifndef ZODIAC_H
#define ZODIAC_H //Оголошення змінної препроцесору
#include <string>
std::string fzodiac(const unsigned &, const unsigned &); //Оголошення
//власної функції з параметрами посиланнями на константи типу unsigned
#endif
```

C++

Завдання до комп'ютерного практикуму №4

Частина 1.

1. Обчислити та запам'ятати значення $z_i = F(x_i, y_i)$ в n різних точках (x_i, y_i) , де $n > 16$; x_i та y_i – довільні додатні числа;

$$F(x_i, y_i) = \sin^2(x_i^2 \cdot y_i) + \cos^3(x_i \cdot y_i^3) - \sqrt{\lg^3 x_i^2};$$

2. Визначити втрату тиску P охолоджуючої води, яка протікає через свинцевий змійовик, який має 40 витків діаметром $D = 0,5$ м. Внутрішній діаметр свинцевої труби $d = 0,012$ м. Середня температура охолоджуючої води 20°C , а середня температура внутрішньої стінки труби 40°C . Витрата охолоджуючої води $G = 0,1$ кг/с.

$$P = L \cdot \frac{l}{d} \cdot \frac{W^2}{2} \cdot p \cdot f \cdot f_1,$$

де $L = 0.0335$ – коефіцієнт тертя;

$l = 3.14 \cdot D \cdot n$ – загальна довжина труби ($n = 40$), м;

$W = \frac{4 \cdot G}{3.14 \cdot p \cdot d^2}$ – швидкість руху рідини, м/с ($G = 998$ кг/м³ – щільність води при 20°C);

f, f_1 – поправкові множини для неізотермічного потоку рідини у непрямих трубах ($f = 0,85$; $f_1 = 1 + 3,54d/D$).

3. Обчислити та запам'ятати значення $z_i = F(x_i, y_i)$ в n різних точках (x_i, y_i) , де $n > 16$; x_i – довільні додатні числа; $y_i = \lg^2 x_i^3 + (i-1)/10$

$$F(x_i, y_i) = \lg^2(x_i - y_i) \cdot (x_i^2 + 0,5 \cdot y_i).$$

4. Знайти коефіцієнт тепловіддачі L при конденсації пару на поверхні труби довжиною $H=2$ м, яка утворює з горизонтальною поверхнею кут $b = \frac{\pi}{3}$. Температура стінки $t_{\text{ст}} = 210^\circ\text{C}$, температура пари $t = 220^\circ\text{C}$.

$$L_b = L \sqrt{\sin b},$$

де L – коефіцієнт тепловіддачі для вертикальної труби, який розраховується за формулою:

$$L = 3 \cdot 10^{-3} \sqrt{\frac{H(t - t_{\text{ст}}) \cdot l^3 \cdot p^2 \cdot g}{r \cdot m^3}}$$

тут $l = 0,652$ Вт/(м*К) – теплопровідність конденсату;

$p = 846$ кг/м³ – щільність конденсату;

$g = 9,81$ м/с² – прискорення вільного падіння;

$r = 1,88 \cdot 10^6$ Дж/кг – теплота пароутворення;

$m = 1,29 \cdot 10^{-4}$ н*с/м² – динамічна в'язкість конденсату.

5. Обчислити значення:

$$y_{ij} = z_i + \sqrt{z_i^2 + a_j^3 \cdot z_i^{0,3}}; \quad z_i = e^{-5 \cdot x_i}.$$

Прийняти $x_i = 0,1; 0,4; 0,8; 1,2$; $a_j = 0,2; 0,5; 0,9$.

6. Розрахувати константу швидкості реакції між йодистим метилом і диметил-п-толундіном у розчині нітробензолу.

$$\frac{\frac{x + x_p}{a}}{a \cdot t \cdot [1 - (\frac{x_p}{a})^2]} \cdot \ln \left| \frac{\frac{x}{a} \left(1 - \frac{x + x_p}{a^2}\right) - 1}{a^2 \cdot \left(\frac{x_p}{a} - \frac{x}{a}\right)} \right|$$

де $a = 0,06$ – початкова концентрація;

$x = 0,01246$ – рівноважна концентрація;

$x_p = 0,00675$ – поточна концентрація;

$t = 12$.

7. Обчислити значення:

$$y_{ij} = (z_i^2 + 0,33) / \operatorname{tg} z_i - b \cdot \sqrt{z_i^2 - a_j}; \quad z_i = e^{-1/x_i} + x_i^2.$$

Прийняти $x_i = 1,3; 1,5; 1,7; 1,9$; $a_j = 0,33; 1,12; 3,72; 5,16$; $b = 3,68$.

8. Розрахувати в'язкість діоксида сірки при 300°C і атмосферному тиску, використавши формулу:

$$\mu = 6,3 \cdot 10^{-4} \cdot \frac{m^{1/2} \cdot P_{\text{кр}}^{2/3}}{T_{\text{кр}}^{1/6}} \cdot \frac{T_{\text{прив}}^{3/2}}{T_{\text{прив}} \cdot 0,8}$$

де $m = 64$ – молекулярна вага SO_2 ;

$T_{\text{кр}} = 430^\circ\text{C}$ – критична температура;

$P_{\text{кр}} = 77,7$ атм – критичний тиск;

$T_{\text{прив}} = (300 + 273)/430$ – приведена температура.

Надрукувати значення μ та $\sqrt{\mu}$.

9. Обчислити значення:

$$y_{ij} = z_i^3 - (z_i + a_j \cdot b) / (a_j^{0,4} + z_i^2); \quad z_i = e^{-1/x_i} + x_i \cdot b \cdot \cos x_i.$$

Потім знайти суму всіх отриманих значень функції y_{ij} .

Прийняти $x_i = 1,3; 1,5; 1,8; 2,0$; $a_j = 0,7; 0,85; 0,9$; $b = 5,75$.

10. Рідина витікає із прямої посудини 1 до посудини 2 через круглий отвір $d = 2$ см = $0,02$ м. Різниця рівнів води в початковий момент $H = 2$ м. Площина поперечних перерізів посудини $S_1 = 8\text{ м}^2$, $S_2 = 6\text{ м}^2$. Знайти час, необхідний для того, щоб різниця рівнів зменшилася до 1 м, для цього використати формулу:

$$t = \frac{8 \cdot S_1 \cdot S_2}{c \cdot 3,14 \cdot d^2 (S_1 + S_2) \cdot \sqrt{2 \cdot g}} (\sqrt{H_0} - \sqrt{H})$$

де $c = 0,62$ – коефіцієнт витрат; $g = 9,81$
надрукувати значення $t, e^c, t^2, \sin^2 d$.

11. Обчислити та запам'ятати значення $z_i = F(x_i, y_i)$ в n різних точках (x_i, y_i) , а потім знайти добуток всіх отриманих значень функції. Прийняти $n > 16$; $x_{i+1} = x_i + \delta x$ ($x_0 = 1.3$; $\delta x = 0.2$); $y_i = x_i^2$;

$$F(x_i, y_i) = \lg^2(x_i \cdot y_i) + \frac{\sqrt{x_i^2 + y_i^2}}{x_i + y_i};$$

12. Розрахувати коефіцієнт ефективності, використовуючи формулу:

$$\Phi_s = \frac{r_s}{3} \sqrt{\frac{W_p \cdot p_k}{n \cdot D_0 \cdot (C_s - C_0)}}$$

де $r_s = 0,317$;
 $W_p = 4,85 \cdot 10^{-4}$;
 $p_k = 1$;
 $n = 0,93$;
 $D_0 = 0,08$;
 $C_s = 1,6 \cdot 10^{-4}$;
 $C_0 = 0$.

Надрукувати значення $\Phi_s, e^n, \sin_s r$.

13. Обчислити та запам'ятати значення $z_i = F(x_i, y_i)$ в n різних точках (x_i, y_i) , де $n > 15$; $x_{i+1} = x_i + \delta x$ ($x_0 = 1,2$; $\delta x = 0,3$); y_i – довільні додатні числа;

$$F(x_i, y_i) = e^{-1/x} + x^y - y^x$$

14. Визначити константу швидкості реакції окислення оксиду азоту, використовуючи формулу:

$$K = \frac{1}{t} \cdot \frac{1}{(b-a)^2} \cdot \left[\frac{(b-a) \cdot x}{(a-x) \cdot a} + \ln \frac{(a-x) \cdot b}{(b-x) \cdot a} \right]$$

де $b=7,06$;
 $a=5,4$;
 $t=11$;
 $x=2,25$.

15. Знайти діаметр корпусу випарювального апарату з природною циркуляцією розчину, використовуючи формулу:

$$D = \sqrt{\frac{1,27 \cdot n \cdot t^2 \cdot \sin L}{f} + (d \cdot 2 \cdot t)^2}$$

де $n = 1210$ – кількість трубок;

$t = 0,05$ м;

$d = 0,4$ м;

$f = 0,85$ м;

$L = 1,047$ рад.

16. Обчислити та запам'ятати значення $z_i = F(x_i, y_i)$ в n різних точках (x_i, y_i) , а потім знайти суму перших п'яти значень та добуток двох останніх.

$$F(x_i, y_i) = e^{-2/x} + 0,8 \cdot x^y - 1,7 \cdot y^x.$$

Прийняти $n > 18$; $x_{i+1} = x_i + \delta x$ ($x_0 = 1.2$; $\delta x = 0.55$); y_i – довільні додатні числа;

17. Знайти коефіцієнт дифузії молекул ізобутану через шар зернин оксиду алюмінію, використовуючи формулу:

$$D = \frac{-R \cdot T \cdot r \cdot g_a}{0,023 \cdot P \cdot (x_2 - x_1)}$$

де $R = 82,298$ – універсальна газова стала;

$T = 0,31$;

$r = 0,5$;

$g_a = 0,08 \cdot 10^{-6}$;

$P = 5$;

$x_2 = 0$;

$x_1 = 1$.

Надрукувати значення D, \sqrt{D}, e^T .

18. Обчислити значення:

$$y_i = a_i^{b_i} + b_i^{a_i}; \quad a_i = e^{-1/x_i} + x_i \cdot C; \quad b_i = \cos^2 x_i + x_i^{0,3}.$$

Прийняти $x_i = 0,5; 0,8; 0,9; 1,2; 1,3; 1,5; 1,7; 1,8; 1,9; 2,1; 2,3; 2,4$; $C = 1,86$.

19. Визначити коефіцієнт дифузії двооксиду сірки у повітрі $t = 20^\circ \text{C}$ та $P = 1$ атм. Використати формулу:

$$D = 4,3 \cdot 10^{-3} \cdot \frac{T^{3/2}}{P \cdot (V_A^{1/3} - V_B^{1/3})^2} \cdot \sqrt{\frac{1}{M_A} + \frac{1}{M_B}}$$

де $V_A = V_{\text{SO}_2} = 44,8$ см³/моль;

$V_B = V_{\text{BO}_3} = 29,9$ см³/моль;

$T = 273 + t$;

$M_A = 64$;

$$M_B = 28,95.$$

20. Обчислити та запам'ятати значення $z_i = F(x_i, y_i)$ в n різних точках (x_i, y_i) , де $n > 18$; x_i – довільні додатні числа; y_i – довільні від'ємні числа.

$$F(x_i, y_i) = 12 \cdot x_i^{0.65} - (1,9 \cdot x_i^2 - 0,85 \cdot y_i^{0.8}) \cdot \lg^2(x_i) / y_i.$$

21. Визначити теплопровідність рідкого гептану при температурі $t = 20^\circ\text{C}$, використовуючи формулу:

$$L = A \cdot C_M \cdot \gamma_{\text{відн}} \cdot \sqrt[3]{\frac{\gamma_{\text{отн}}}{M}}$$

де $A = 1,52$;

$C_M = 51,9 + 0,142 \cdot (t + 30)$, кал/(моль * град) – мольна теплоємність;

$\gamma_{\text{відн}} = 0,675$ – відносна питома вага;

$M = 100$;

$C = C_M / 100$ – питома теплоємність.

22. Обчислити та запам'ятати значення:

$$y_{ij} = (A_i \cdot \sin B_j + B_j \cdot \cos A_i) \cdot (A_i^{0.5} + \sqrt[3]{B_j});$$

$$A_i = \cos^2 x_i + x_i^3; B_j = \sin^2 a_j + a_j^{0.5}.$$

Прийняти $x_i = 1,2; 0,4; 1,3; 0,8$; $a_j = 0,6; 0,9; 1,9$.

23. Знайти тривалість адсорбції суміші парів етилового спирту та діетилового ефіру шаром активованого вугілля висотою $H = 1\text{м}$, використовуючи формулу:

$$t = \frac{a_0}{w \cdot C_0} \cdot \left[H - \frac{w}{b} \cdot \left(\ln \frac{C_0}{C} - 1 \right) \right]$$

де $a_0 = 100 \text{ кг/м}^3$ – кількість адсорбції речовини, зрівноваженої з концентрацією потоку;

$w = 12 \text{ м/хв}$ – швидкість парогазового потоку;

$C_0 = 0,072 \text{ кг/м}^3$ – початкова концентрація;

$C = 0,001 \text{ кг/м}^3$ – середня концентрація на виході;

$b = 60 \cdot 7,4 \text{ хв}^{-1}$ – коефіцієнт масо передачі.

24. Обчислити та запам'ятати значення $z_i = F(x_i, y_i)$ в n різних точках (x_i, y_i) , а потім знайти суму перших п'яти значень та добуток двох останніх. Прийняти $n > 16$; $x_{i+1} = x_i + \delta x$ ($x_0 = 1.5$; $\delta x = 0.65$); y_i – довільні додатні числа;

$$F(x_i, y_i) = 0,8 \cdot x^y - 1,7 \cdot y^x + \ln \frac{x}{y}.$$

25. Обчислити та запам'ятати значення $z_i = F(x_i, y_i)$ в n різних точках (x_i, y_i) , де $n > 16$; $x = x_i \cdot \delta x$ ($x = 1,8$; $\delta x = 1,2$); y_i – довільні додатні числа ($y_i < 1$).

$$F(x_i, y_i) = \frac{\sqrt{1-y_i}}{x_i + \operatorname{tg} x_i} \cdot (1 + e^{-x_i \cdot y_i}).$$

26. Обчислити значення:

$$y_{ij} = z_i^{0,3} + z_i \cdot a_j; \quad z_i = x_i^2 + \ln x_i.$$

Прийняти $x_i = 1,2; 1,6; 1,9; 1,95; a_j = 0,9; 0,95; 0,99; b=1,68$.

27. Обчислити значення:

$$y_{ij} = z_i + (a_j \cdot z_i \cdot b + b^{0,3}) / \sqrt{z_i^2 + 1}; \quad z_i = e^{-1/x_i} \cdot i + x_i \cdot b.$$

Прийняти $x_i = 1,9; 1,5; 1,4; 1,2; a_j = 0,2; 1,1; 1,4$.

28. Обчислити значення:

$$y_{ij} = z_i^3 - (z_i + a_j \cdot b) / (a_j^{0,3} + z_i^2); \quad z_i = e^{-2/x_i} + x_i \cdot b \cdot \cos(x_i).$$

Прийняти $x_i = 1,2; 1,6; 1,9; 2,2; a_j = 0,75; 0,8; 0,9; 0,95; b = 6,74$.

29. Обчислити та запам'ятати значення $z_i = F(x_i, y_i)$ в n різних точках (x_i, y_i) , де $n > 18; x_{i+1} = x_i + e^{\delta x}$ ($x_0 = 2; \delta x = 0,1$); $y_{i+1} = dy^*(1 - y_i)$ ($y_0 = 3; dy = -2$).

$$F(x_i, y_i) = 15^{2x-3y} + \sin^2(x^3 + y^4) + x^2 y^3.$$

30. Обчислити значення:

$$y_{ij} = 0,88 \cdot z_i + \sqrt{\alpha \cdot z_i^2 + a_j^{2,5} \cdot z_i^{0,45}}; \quad z_i = \lg(3,8 \cdot x_i).$$

Прийняти $x_i = 1,1; 1,4; 1,8; 2,2; 2,5; 3,1; a_j = 0,2; 0,5; 0,9$.

Частина 2.

1. Обчислити та запам'ятати значення $z_i = F(x_i, y_i)$ в n різних точках (x_i, y_i) , а потім знайти суму всіх отриманих значень функції. Прийняти $n = 14$; $x_{i+1} = x_i + \delta x$ ($x_0 = 2,8$; $\delta x = 2,1$); $y_{i+1} = y_i + \delta y$ ($y_0 = 1,7$; $\delta y = 0,3$).

$$F(x_i, y_i) = \left[\sin(x_i^2) - \cos(y_i^2) \right] \frac{\sqrt[3]{2x_i - y_i}}{\cos(x_i)}.$$

2. Обчислити значення:

$$y_{ij} = z_i \cdot \cos^2 a_j + \sqrt{z_i^3 + a_j \cdot z_i}; \quad z_i = \cos(x_i + 1).$$

Прийняти $x_i = 0,3; 0,8; 1,2$; $a_j = 1,1; 1,5; 1,9; 2,2$.

3. Обчислити значення:

$$y_{ij} = z_i^2 + \sqrt{z_i^2 - a_j}; \quad \text{где } z_i = e^{1/x_i} + x_i^2.$$

Прийняти $x_i = 1,2; 1,8; 1,6; 1,4$; $a_j = 0,4; 0,5; 0,8$.

4. Обчислити та запам'ятати значення $z_i = F(x_i, y_i)$ в n різних точках (x_i, y_i) , де $n = 14$; $x_{i+1} = x_i + \delta x$ ($x_0 = 1,2$; $\delta x = 0,25$); $y_i = x_i^2$.

$$F(x_i, y_i) = \lg^2(x_i \cdot y_i) + \frac{\sqrt{x_i^2 + y_i^2}}{x_i + y_i}.$$

5. Обчислити та запам'ятати значення $z_i = F(x_i, y_i)$ в n різних точках (x_i, y_i) , де $n = 15$; $x_{i+1} = x_i + \delta x$ ($x_0 = 4,2$; $\delta x = 1,8$); $y_{i+1} = y_i + \delta y$ ($y_0 = 1,82$; $\delta y = 0,33$).

$$F(x_i, y_i) = \left[\ln(x_i^2) - \ln(y_i^2) \right] \frac{\sqrt[3]{x_i - y_i}}{\operatorname{ctg}(x_i)};$$

6. Обчислити та запам'ятати значення $z_i = F(x_i, y_i)$ в n різних точках (x_i, y_i) , де $n = 16$; x_i та y_i – довільні додатні числа.

$$F(x_i, y_i) = \sin^2(x_i^2 \cdot y_i) + \cos^3(x_i \cdot y_i^3) - \sqrt{\lg^3 x_i^2}.$$

7. Обчислити та запам'ятати значення $z_i = F(x_i, y_i)$ в n різних точках (x_i, y_i) , де $n = 16$; x_i – довільні додатні числа; $y_i = \lg^2 x_i^3 + (i-1)/10$.

$$F(x_i, y_i) = \operatorname{tg}^2(x_i - y_i) \cdot (x_i^2 + 0,5 \cdot y_i).$$

8. Обчислити значення:

$$y_{ij} = z_i + \sqrt{z_i^2 + a_j^3 \cdot z_i^{0,3}}; \quad z_i = e^{-5 \cdot x_i}.$$

Прийняти $x_i = 0,1; 0,4; 0,8; 1,2$; $a_j = 0,2; 0,5; 0,9$.

9. Обчислити значення:

$$y_{ij} = (z_i^2 + 0,33) / \operatorname{tg} z_i - b \cdot \sqrt{|z_i^2 - a_j|}; \quad z_i = e^{-1/x_i} + x_i^2.$$

Прийняти $x_i = 1,3; 1,5; 1,7; 1,9$; $a_j = 0,33; 1,12; 3,72; 5,16$; $b=3,68$.

10. Обчислити значення:

$$y_{ij} = z_i^3 - (z_i + a_j \cdot b) / (a_j^{0,4} + z_i^2); \quad z_i = e^{-1/x_i} + x_i \cdot b \cdot \cos x_i.$$

Потім знайти суму всіх отриманих значень функції y_{ij} .

Прийняти $x_i = 1,3; 1,5; 1,8; 2,0$; $a_j = 0,7; 0,85; 0,9$; $b = 5,75$.

11. Обчислити та запам'ятати значення $z_i = F(x_i, y_i)$ в n різних точках (x_i, y_i) , а потім знайти добуток всіх отриманих значень функції.

$$F(x_i, y_i) = \lg^2(x_i \cdot y_i) + \frac{\sqrt{x_i^2 + y_i^2}}{x_i + y_i}.$$

Прийняти $n = 16$; $x_{i+1} = x_i + \delta x$ ($x_0 = 1,3$; $\delta x = 0,2$); $y_i = x_i^2$.

12. Обчислити та запам'ятати значення $z_i = F(x_i, y_i)$ в n різних точках (x_i, y_i) :

$$F(x_i, y_i) = e^{-1/x} + x^y - y^x,$$

де $n = 15$; $x_{i+1} = x_i + \delta x$ ($x_0 = 1,2$; $\delta x = 0,3$); y_i – довільні додатні числа.

13. Обчислити та запам'ятати значення:

$$y_{ij} = A_i^2 + A_i \cdot B_i \cdot C_j - B_i^2 \cdot \sqrt{A_i + B_i}; \quad A_i = \sin^2 x_i; \quad B_i = e^{1/x_i} + 3,$$

а потім знайти суму всіх отриманих значень функції y_{ij} .

Прийняти $x_i = 1,8; 1,4; 1,1; 1,5$; $C_j = 3,11; 2,21; 2,87$.

14. Обчислити та запам'ятати значення $z_i = F(x_i, y_i)$ в n різних точках (x_i, y_i) , а потім знайти суму перших п'яти значень та добуток двох останніх.

$$F(x_i, y_i) = e^{-2/x} + 0,8 \cdot x^y - 1,7 \cdot y^x.$$

Прийняти $n = 18$; $x_{i+1} = x_i + \delta x$ ($x_0 = 1,2$; $\delta x = 0,55$); y_i – довільні додатні числа.

15. Обчислити значення:

$$y_i = a_i^{b_i} + b_i^{a_i}; \quad a_i = e^{-1/x_i} + x_i \cdot C; \quad b_i = \cos^2 x_i + x_i^{0,3}.$$

Прийняти $x_i = 0,5; 0,8; 0,9; 1,2; 1,3; 1,5; 1,7; 1,8; 1,9; 2,1; 2,3; 2,4$; $C=1,86$.

16. Обчислити та запам'ятати значення $z_i = F(x_i, y_i)$ в n різних точках (x_i, y_i) ,

$$F(x_i, y_i) = 12 \cdot x_i^{0,65} - (1,9 \cdot x_i^2 - 0,85 \cdot y_i^{0,8}) \cdot \operatorname{tg}^2(x_i) / y_i.$$

де $n = 18$; x_i – довільні додатні числа; y_i – довільні від'ємні числа.

17. Обчислити та запам'ятати значення:

$$y_{ij} = (A_i \cdot \sin B_j + B_j \cdot \cos A_i) \cdot (A_i^{0,5} + \sqrt[3]{B_j});$$

$$A_i = \cos^2 x_i + x_i^3; B_j = \sin^2 a_j + a_j^{0,5}.$$

Прийняти $x_i = 1,2; 0,4; 1,3; 0,8; a_j = 0,6; 0,9; 1,9$.

18. Обчислити значення:

$$y_{ij} = z_i + (a_j \cdot z_i \cdot b + b^{0,3}) / \sqrt{z_i^2 + 1}; \quad z_i = e^{-1/x_i} \cdot i + x_i \cdot b.$$

Прийняти $x_i = 1,2; 1,6; 1,9; 1,95; a_j = 0,9; 0,95; 0,99; b = 1,68$.

19. Обчислити значення:

$$y_{ij} = z_i^{0,3} + z_i \cdot a_j; \quad z_i = x_i^2 + \ln x_i.$$

Прийняти $x_i = 1,9; 1,5; 1,4; 1,2; a_j = 0,2; 1,1; 1,4$.

20. Обчислити та запам'ятати значення $z_i = F(x_i, y_i)$ в n різних точках (x_i, y_i) , а потім знайти суму перших п'яти значень та добуток двох останніх.

$$F(x_i, y_i) = 0,8 \cdot x^y - 1,7 \cdot y^x + \ln \frac{x}{y}.$$

Прийняти $n = 16; x_{i+1} = x_i + \delta x$ ($x_0 = 1,5; \delta x = 0,65$); y_i – довільні додатні числа.

21. Обчислити та запам'ятати значення $z_i = F(x_i, y_i)$ в n різних точках (x_i, y_i) ,

$$F(x_i, y_i) = \frac{\sqrt{1 - y_i}}{x_i + \operatorname{tg} x_i} \cdot \left(1 + e^{-x_i \cdot y_i} \right).$$

де $n = 16; x = x_i \cdot \delta x$ ($x = 1,8; \delta x = 1,2$); y_i – довільні додатні числа ($y_i < 1$).

22. Обчислити значення:

$$y_{ij} = z_i^3 - (z_i + a_j \cdot b) / (a_j^{0,3} + z_i^2); \quad z_i = e^{-2/x_i} + x_i \cdot b \cdot \cos(x_i).$$

Прийняти $x_i = 1,2; 1,6; 1,9; 2,2; a_j = 0,75; 0,8; 0,9; 0,95; b = 6,74$.

23. Обчислити та запам'ятати значення $z_i = F(x_i, y_i)$ в n різних точках (x_i, y_i) ,

$$F(x_i, y_i) = 15^{2x-3y} + \sin^2(x^3 + y^4) + x^2 y^3,$$

де $n = 18; x_{i+1} = x_i + e^{\delta x}$ ($x_0 = 2; \delta x = 0,1$); $y_{i+1} = dy \cdot (1 - y_i)$ ($y_0 = 3; dy = -2$).

24. Обчислити значення:

$$y_{ij} = 0,88 \cdot z_i + \sqrt{\alpha \cdot z_i^2 + a_j^{2,5} \cdot z_i^{0,45}}; \quad z_i = \lg(3,8 \cdot x_i).$$

Прийняти $x_i = 1,1; 1,4; 1,8; 2,2; 2,5; 3,1; a_j = 0,2; 0,5; 0,9$.

25. Обчислити та запам'ятати значення:

$$y_{ij} = A_i^{2,2} + 1,5 \cdot A_i \cdot B_i \cdot C_j - B_i^{1,5} \cdot \sqrt{A_i + 2B_i}; \quad A_i = 1,8 \cdot \cos^3 x_i; \quad B_i = e^{\frac{2}{x_i}} + 2,6,$$

а потім знайти суму всіх отриманих значень функції y_{ij} .

Прийняти $x_i = 2,2; 1,9; 1,7; 1,4; 1,2; C_j = 2,31; 2,86; 3,57$.

26. Обчислити значення:

$$y_i = a_i^{2b_i} + b_i^{3a_i}; \quad a_i = e^{2,5/x_i} + 3x_i \cdot C; \quad b_i = \sin^2 x_i + x_i^{0,5}.$$

Прийняти $x_i = 0,3; 0,7; 0,9; 1,1; 1,2; 1,4; 1,7; 1,8; 1,9; 2,2; 2,5; 2,7; C = 2,15$.

27. Обчислити значення:

$$y_{ij} = z_i \cdot \cos^2 a_j + \sqrt{z_i^3 + a_j \cdot z_i}; \quad z_i = \cos(x_i + 1).$$

Прийняти $x_i = 0,3; 0,8; 1,2; a_j = 1,1; 1,5; 1,9; 2,2$.

28. Обчислити значення:

$$y_{ij} = z_i^2 + \sqrt{z_i^2 - a_j}; \quad \text{де } z_i = e^{\frac{1}{x_i}} + x_i^2.$$

Прийняти $x_i = 1,2; 1,8; 1,6; 1,4; a_j = 0,4; 0,5; 0,8$.

29. Обчислити та запам'ятати значення $z_i = F(x_i, y_i)$ в n різних точках (x_i, y_i) , де $n > 14$; $x_{i+1} = x_i + \delta x$ ($x_0 = 1,2$; $\delta x = 0,25$); $y_i = x_i^2$.

$$F(x_i, y_i) = \lg^2(x_i \cdot y_i) + \frac{\sqrt{x_i^2 + y_i^2}}{x_i + y_i}.$$

30. Обчислити та запам'ятати значення $z_i = F(x_i, y_i)$ в n різних точках (x_i, y_i) , де $n > 20$; $x_{i+1} = x_i + \delta x$ ($x_0 = 3,2$; $\delta x = 1,2$); $y_{i+1} = y_i + \delta y$ ($y_0 = 1,32$; $\delta y = 0,33$).

$$F(x_i, y_i) = \left[\ln(x_i^2 - 1) - \ln(y_i^2 + 1) \right] \frac{\sqrt{x_i - 3y_i}}{\lg(x_i + 1)}.$$

Контрольні питання

- 1) Перерахуйте арифметичні типи мови C++ та вкажіть, які значення вони можуть зберігати.
- 2) Що таке знакові та беззнакові цілочиселні типи? Перерахуйте беззнакові цілочисельні типи.
- 3) Розкажіть що таке автоматичне перетворення типів та за якими правилами воно виконується.
- 4) Що таке змінна? З чого складається визначення змінної?
- 5) Що таке ініціалізація та для чого вона потрібна?
- 6) Що таке ідентифікатори? Які існують правила до створення ідентифікатора?
- 7) Що таке область видимості? Яка область видимості у об'єктів класу?
- 8) Що таке складений тип? Які складені типи ви знаєте?
- 9) Що таке посилання? Які правила використання та обмеження існують для посилань?
- 10) Що таке покажчик? Які правила використання існують для покажчиків? Як звернутись до значення покажчика?
- 11) Що таке специфікатор типу `auto`? Коли і як його слід використовувати?
- 12) Що таке специфікатор типу `decltype`? Коли і як його слід використовувати?
- 13) Що таке власна структура даних або клас? З чого складається визначення класу?
- 14) Що таке змінні-члени? Що таке внутрішньокласовий ініціалізатор?
- 15) Для чого потрібні файли заголовку? Як підключити заголовок до програми? Як організувати захист заголовку?
- 16) Що таке змінні препроцесора? Які правила створення імен змінних препроцесору та файлів заголовку загальноприйняті? Як працює директива `#include`?
- 17) Які таблиці кодування використовуються у Windows? У чому їх відмінності?
- 18) Що потрібно для коректного відображення кирилиці у консольному додатку? Для чого використовують файл заголовку `windows.h`? Для чого потрібні функції `SetConsoleCP()` і `SetConsoleOutputCP()`?

Комп'ютерний практикум №5

Типи `string` та масиви

Мета: ознайомитись із бібліотечним типом `string`: операції з рядками; робота з символами рядка. Засвоїти використання *оператору індексування* для роботи з *структурами даних*. Ознайомитись із використанням *ітераторів* для типу `string`. Вивчити роботу з *масивами*: *визначення та ініціалізація*; доступ до елементів; застосування *показчиків*; *динамічні масиви*.

Завдання: створити консольний додаток для роботи з рядковим текстом, а також проведення розрахунків за допомогою одномірного масиву для вирішення поставленої задачі згідно отриманого варіанту завдання.

Загальні вимоги.

- 1) Створити консольний додаток з ім'ям виду *PrizvischeKP5*.
- 2) Програмний код модуля має бути чітко структурований.
- 3) Імена об'єктів мають нести сенсові навантаження.
- 4) Програмний код має супроводжуватись коментарями в тексті програми.

Вимоги до виконання.

Частина 1.

- 1) Запрограмувати роботу з рядковим типом `string`.
- 2) Звернення до елементів рядку організувати з використанням *ітераторів*.
- 3) Введення вхідних даних організувати в режимі діалогу з користувачем.
- 4) Завдання для обчислень, вхідні дані та результати розрахунків вивести у зовнішній текстовий файл з ім'ям виду *ПрізвищеКП5_з1_OUT*.
- 5) Виведення даних у вихідний файл організувати в режимі додавання в кінець файлу.

Частина 2.

- 1) Запрограмувати роботу з одномірним *динамічним масивом* за допомогою вбудованого типу *масив*.
- 2) Звернення до елементів *масиву* організувати з використанням *показчиків*.
- 3) Введення вхідних даних організувати в режимі діалогу з користувачем.
- 4) Завдання для обчислень, вхідні дані та результати розрахунків вивести у зовнішній текстовий файл з ім'ям виду *ПрізвищеКП5_з2_OUT*.

- 5) Виведення даних у вихідних файл організувати в режимі додавання в кінець файлу.

Теоретичні відомості

Тема 5.1. Бібліотечний тип string

Рядок (string) – це послідовність символів змінної довжини. Для використання типу string, необхідно включити до коду заголовок string. Оскільки тип string належить бібліотеці, він визначений у просторі імен std.

```
#include <string>
using std::string;
```

5.1.1 Ініціалізація об'єктів класу string

Кожен клас визначає, як можуть бути ініціалізовані об'єкти його типу. Перелік найбільш розповсюджених ініціалізаторів об'єктів типу string наведений в табл. 5.1.1.

Таблиця 5.1.1 – Способи ініціалізації об'єкту класу string

Ініціалізація	Роз'яснення
string s1	Ініціалізація за замовчуванням; s1 – пустий рядок
string s2(s1)	s2 - копія s1
string s2 = s1	Еквівалент s2(s1), s2 - копія s1
string s3("value")	s3 копія рядкового літерала, нульовий символ не включений
string s3 = "value"	Еквівалент s3("value"), s3 копія рядкового літерала
string s4(n, 'c')	Ініціалізація змінної s4 символом c в кількості n штук

Коли змінна ініціалізується з використанням знаку =, компілятор копіює ініціалізуючий об'єкт в створюваний об'єкт, тобто відбувається *ініціалізація копією* (copy initialization). В іншому випадку, без знаку = та з використанням круглих дужок відбувається *пряма ініціалізація* (direct initialization).

5.1.2 Операції класу string

Окрім способів створення та ініціалізації об'єктів клас визначає також операції, які можна виконувати з об'єктами класу. Найбільш розповсюджені операції класу string наведені в табл. 5.1.2.

Таблиця 5.1.2 – Операції класу string

Операція	Роз'яснення
<code>os << s</code>	Виводить рядок <code>s</code> у потік виведення <code>os</code> . Повертає потік <code>os</code>
<code>is >> s</code>	Читає розділений пробілами рядок <code>s</code> з потоку <code>is</code> . Повертає потік <code>is</code>
<code>getline(is, s)</code>	Читає рядок введення з потоку <code>is</code> у змінну <code>s</code> . Повертає потік <code>is</code>
<code>copy()</code>	Копіює послідовність символів з рядка
<code>find()</code>	Знаходить вміст у рядку
<code>rfind()</code>	Знаходить останню появу вмісту в рядку
<code>find_first_of()</code>	Знаходить символ у рядку
<code>find_last_of()</code>	Знаходить символ у рядку з кінця
<code>find_first_not_of()</code>	Знаходить відсутність символу в рядку
<code>find_last_not_of()</code>	Знаходить невідповідний символ у рядку з кінця
<code>substr()</code>	Створює підрядок
<code>compare()</code>	Порівнює рядки
<code>s1 + s2</code>	Повертає рядок, що складається з вмісту рядків <code>s1</code> та <code>s2</code>
<code>s1 = s2</code>	Замінює символи рядку <code>s1</code> копією вмісту рядку <code>s2</code>
<code>s1 == s2</code>	Рядки <code>s1</code> та <code>s2</code> рівні, якщо містять однакові символи. Регістр символів враховується
<code>s1 != s2</code>	Рядки <code>s1</code> та <code>s2</code> не рівні, якщо хоча б один символ чи регістр символу не співпадає
<code><, <=, >, >=</code>	Порівняння залежить від регістру і покладається на алфавітний порядок символів

5.1.3 Функції для роботи з розміром рядка string

Для доступу до розміру рядка у класі `string` та можливості зміни довжини рядка у C++ передбачені певні функції-члени та оператори, які наведені в табл. 5.1.3.

Таблиця 5.1.3 – Функції для роботи з розміром рядка string

Функція	Роз'яснення
<code>s.empty()</code>	Повертає значення <code>true</code> , якщо рядок <code>s</code> пустий. Інакше повертає значення <code>false</code>
<code>s.size()</code>	Повертає довжину рядка, тобто кількість символів в ньому
<code>s.clear()</code>	Стирає вміст рядка, який стає порожнім (довжиною 0 символів)
<code>s.resize(n)</code>	Встановлює довжину рядка рівною <code>n</code> символів
<code>s.length()</code>	Повертає кількість символів у рядку

Використання функцій для роботи з розміром рядка `string` наведене у наступному прикладі.

```
#include "stdafx.h"
#include <iostream>
#include <string>
```



```

#include <windows.h>
#include <locale>

int main()
{
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);

    std::string рядок("Операції з розміром рядка");
    std::cout << "Вихідний рядок: " << рядок << "\n";
    std::cout << "Розмір рядка: " << рядок.size() << "\n";
    std::cout << "Довжина рядка: " << рядок.length() << "\n";
    рядок.resize(19);
    std::cout << "Зміна розміру рядка: " << рядок << "\n";
    рядок.clear();
    std::cout << "Очищення рядка: " << рядок << "\n";
    if (рядок.empty())
    {
        std::cout << "Рядок пустий!\n";
    }
    return 0;
}

```

Результат роботи прикладу буде наступним:

Вихідний рядок: Операції з розміром рядка
 Розмір рядка: 25
 Довжина рядка: 25
 Зміна розміру рядка: Операції з розміром
 Очищення рядка:
 Рядок пустий!

5.1.4 Доступ до елементів рядка string

Для доступу до певних елементів рядка у класі `string` передбачені певні функції-члени та оператори, які наведені в табл. 5.1.4.

Таблиця 5.1.4 – Доступ до елементів у класі `string`

Операція	Роз'яснення
<code>s[n]</code>	Повертає посилання на символ у позиції <code>n</code> рядку <code>s</code> ; позиції відраховуються від 0
<code>s.at(i)</code>	Повертає посилання на символ у позиції <code>i</code> у рядку
<code>s.back()</code>	Повертає посилання на останній символ в рядку
<code>s.front()</code>	Повертає посилання на перший символ в рядку

5.1.5 Зміна елементів рядка string

Для зміни певних елементів рядка у класі `string` передбачені набір функцій-членів та операторів, які наведені в табл. 5.1.5.

Таблиця 5.1.5 – Зміна рядка класу `string`

Операція	Роз'яснення
<code>operator+=</code>	Додає в кінець рядка символи чи інший рядок
<code>append()</code>	Додає в кінець рядка символи, інший рядок чи його частину
<code>push_back()</code>	Додає символ у кінець рядка та збільшує його довжину на одиницю
<code>assign()</code>	Призначає вміст для рядка
<code>insert()</code>	Вставляє в рядок символи, інший рядок чи його частину
<code>erase()</code>	Очищає символи в рядку
<code>replace()</code>	Переміщує частину рядка
<code>swap()</code>	Міняє місцями рядкові значення
<code>pop_back()</code>	Видаляє останній символ

Приклади використання деяких перевантажених функцій-членів класу `string` для зміни елементів в рядках наведені у таблиці 5.1.6.

Таблиця 5.1.6 – Приклади застосування функцій-членів класу `string`

Операція	Роз'яснення
<code>s.append(s2)</code>	Додає рядок <code>s2</code> в кінець рядка <code>s</code>
<code>s.append(s2,n,m)</code>	Додає в кінець рядка <code>s</code> символи рядка <code>s2</code> починаючи з позиції <code>n</code> до позиції <code>m</code>
<code>s.append("текст",n)</code>	Додає до рядка <code>s</code> вказаний «текст» у позицію <code>n</code>
<code>s.push_back(c)</code>	Додає у кінець рядка <code>s</code> символ <code>c</code> , збільшуючи довжину рядка на одиницю
<code>s.assign(s2)</code>	Призначає нове значення рядку, замінюючи його поточний зміст
<code>s.insert(n,s2);</code>	Вставляє додаткові символи в рядок <code>s</code> праворуч починаючи з вказаної позиції <code>n</code>
<code>s.insert(n,s2,k,m);</code>	Вставляє у позицію <code>n</code> рядка <code>s</code> символи з рядка <code>s2</code> починаючи з позиції <code>k</code> у кількості <code>m</code>
<code>s.insert(n,"текст для вставки",m)</code>	Вставляю у позицію <code>n</code> рядка <code>s</code> кількість символів <code>m</code> з наведеного текстового рядка
<code>s.insert(n,k,'c')</code>	Вставляє у позицію <code>n</code> рядка <code>s</code> символи 'с' у кількості <code>k</code> штук
<code>s.erase(n,m);</code>	Починаючи з позиції <code>n</code> витирає в рядку <code>s</code> кількість символів, що дорівнює <code>m</code> та відповідно зменшує довжину рядку

5.1.6 Зчитування невизначеної кількості рядків

За допомогою оператора циклу `while` можна організувати алгоритм зчитування невизначеної кількості рядків.


```
#include <iostream>
#include <string>
using std::string;
using std::cout;
using std::cin;
using std::endl;
int main()
{
    string word;
    while (cin >> word) //Читати до кінця файлу
    {
        cout << word << endl; //Відобразити кожне слово з нового рядку
    }
    return 0;
}
```

Умова оператора `while` перевіряє потік після завершення читання. Якщо потік припустимий, тобто не зустрівся символ кінця файлу (його можна задати сполученням клавіш `ctrl + z` під час введення) або неприпустиме значення, виконується тіло циклу `while`. В даному випадку воно виводить прочитане значення на стандартний пристрій виведення. Як тільки зустрічається кінець файлу (або неприпустиме введення), цикл `while` завершується. Аналогічно можна організувати введення невизначеної кількості даних іншого типу, наприклад `int` чи `float`.

5.1.7 Функція `getline()`

Коли не потрібно ігнорувати пробіли у потоці введення, замість оператора `>>` слід використовувати функцію `getline()`. Функція `getline()` отримує потік введення і рядок. Функція читає наданий потік до першого символу нового рядку і зберігає прочитане без символу нового рядку в своєму аргументі типу `string`. Зустрівши символ нового рядку, навіть якщо він перший, функція `getline()` зупиняє читання і завершує роботу.

Подібно оператору введення, функція `getline()` повертає свій аргумент типу `istream`. В результаті функцію `getline()` можна використовувати в умові, як і оператор введення. Щоб програма з попереднього прикладу замість виведення по одному слову окремо вивела весь рядок, потрібно скористатись функцією `getline()`.

```
int main()
{
    string line;
    while (getline(cin, line)) //Читати до кінця файлу
    {
        cout << line << endl; //Вивести весь введений рядок
    }
}
```



```

    }
    return 0;
}

```

5.1.8 Тип `size_type` класу `string`

У класі `string` та деяких інших бібліотечних типах визначені допоміжні типи даних. Ці допоміжні типи дозволяють використовувати бібліотечні типи машинно-незалежним способом. Тип `size_type` – один з таких допоміжних типів. Для використання типу `size_type`, який визначений у класі `string`, застосовується *оператор області видимості* (оператор `::`), який вказує на те, що ім'я `size_type` визначене у класі `string`. Це *беззнаковий цілочисельний тип*, достатньо великий, щоб містити розмір любого рядку. Будь-яка змінна, котра використовується для збереження результату функції `size()` класу `string`, повинна мати тип `string::size_type`.

5.1.9 Функції для роботи з окремими символами рядка

Для можливості виконання певних операцій з окремими символами рядка існує набір бібліотечних функцій, які визначені у заголовку `cctype`. Перелік функцій з їх призначенням міститься в табл. 5.1.7.

Таблиця 5.1.7 – Функції `cctype` для об'єктів типу `string`

Функція	Операція
<code>isalnum(c)</code>	Повертає значення <code>true</code> , якщо <code>c</code> є літерою чи цифрою
<code>isalpha(c)</code>	Повертає значення <code>true</code> , якщо <code>c</code> є літерою
<code>iscntrl(c)</code>	Повертає значення <code>true</code> , якщо <code>c</code> – управляючий символ
<code>isdigit(c)</code>	Повертає значення <code>true</code> , якщо <code>c</code> є цифрою
<code>isgraph(c)</code>	Повертає значення <code>true</code> , якщо <code>c</code> не пробіл, а друкований символ
<code>islower(c)</code>	Повертає значення <code>true</code> , якщо <code>c</code> – символ в нижньому регістрі
<code>isprint(c)</code>	Повертає значення <code>true</code> , якщо <code>c</code> – друкований символ
<code>ispunct(c)</code>	Повертає значення <code>true</code> , якщо <code>c</code> – знак пунктуації
<code>isspace(c)</code>	Повертає значення <code>true</code> , якщо <code>c</code> – символ відступу (пробіл, табуляція, новий рядок і т.п.)
<code>isupper(c)</code>	Повертає значення <code>true</code> , якщо <code>c</code> – символ у верхньому регістрі
<code>isxdigit(c)</code>	Повертає значення <code>true</code> , якщо <code>c</code> є шістнадцятиричною цифрою
<code>tolower(c)</code>	Якщо <code>c</code> прописна літера – переводить її в нижній регістр, інакше залишає без змін
<code>toupper(c)</code>	Якщо <code>c</code> рядкова літера – переводить її у верхній регістр, інакше залишає без змін

Тема 5.2. Доступ до окремих елементів

Існує два способи доступу до окремих символів в рядку: можна використовувати індекс чи ітератор.

5.2.1 Оператор індексування

Оператор індексування (`[]`), як і оператор виклику функції, вважається бінарним оператором. Оператор індексування повинен бути нестатичною функцією-членом, яка приймає один аргумент. Цей аргумент може бути будь-якого типу і визначає необхідний індекс масиву.

Оператор індексування (оператор `[]`) для типу `string` отримує значення типу `string::size_type`, яке позначає позицію символу, до якого потрібен доступ. Оператор повертає посилання на символ у вказаній позиції.

Індексування рядків починається з нуля, першим символом буде `s[0]`, другим `s[1]`, а останнім `s[s.size() - 1]`.

Значення оператора індексування називається індексом (index). Індекс може бути любым виразом, який повертає цілочисельне значення.

5.2.2 Використання ітераторів

Для доступу до символів рядка також можна використовувати *ітератори* (iterator). Окрім рядків і векторів у всіх бібліотечних контейнерів є ітератори, але тільки деякі з них підтримують оператор індексування. Тому ітератори є більш загальним та універсальним інструментом.

Як і вказівники, ітератори забезпечують опосередкований доступ до об'єкту. У випадку ітератора цим об'єктом є елемент в контейнері чи символ у рядку. Ітератор дозволяє вибрати елемент, а також підтримує операції переміщення з одного елементу на інший. Подібно покажчикам, ітератор може бути припустимим та неприпустимим. Припустимий ітератор вказує або на елемент, або на позицію за останнім елементом в контейнері. Всі інші значення ітератора неприпустимі.

5.2.3 Перелік функцій для роботи з ітераторами

На відміну від покажчиків, для отримання ітератора не потрібно використовувати оператор звернення до адреси (`&`). Для цього у типів, які мають ітератори, є члени, котрі повертають ці ітератори. В таблиці 5.2.1 наведений перелік основних функцій-членів для роботи з ітераторами.

Таблиця 5.2.1 – Функції для роботи з ітераторами типу `string`

Функція	Операція
<code>begin</code>	Повертає ітератор до початку
<code>end</code>	Повертає ітератор до кінця
<code>rbegin</code>	Повертає ітератор на початок при переборі у зворотному порядку
<code>rend</code>	Повертає ітератор у кінець при переборі у зворотному порядку
<code>cbegin</code>	Повертає <code>const_iterator</code> на початок
<code>cend</code>	Повертає <code>const_iterator</code> у кінець
<code>crbegin</code>	Повертає <code>const_reverse_iterator</code> на початок при переборі у зворотному порядку
<code>crend</code>	Повертає <code>const_reverse_iterator</code> у кінець при переборі у зворотному порядку

5.2.3.1 Функції `begin()` та `end()`

Типи, які підтримують роботу з ітераторами, мають функції-члени `begin()` та `end()`. Функція-член `begin()` повертає ітератор, який позначає перший елемент (або перший символ), якщо він є. Функція-член `end()` повертає ітератор, який вказує на наступну позицію за кінцем контейнеру (або рядку). Цей ітератор позначає неіснуючий елемент за кінцем контейнеру. Він використовується як індикатор того, що оброблені всі елементи контейнеру. Ітератор, який повертає функція `end()`, називають *ітератором після кінця* (*off-the-end iterator*), або скорочено *ітератором end*. Якщо контейнер пустий, функція `begin()` повертає той самий ітератор, що і функція `end()`.

Тип значень ітератора визначається в контейнері, для якого використовується ітератор. Наприклад, тип ітератора для рядкового типу буде `string::iterator`. Для визначення типу ітератора, як і багатьох інших типів, коли запис типу досить складний або невідомий до отримання значення виразу, зручно використовувати специфікатор `auto`.

```
string s;
//a позначає перший символ рядку s, b – елемент після останнього символу
auto a = s.begin(), b = s.end();
```

5.2.3.2 Використання зворотного ітератора і функцій `rbegin()` та `rend()`

Окрім перебору елементів контейнера по порядку від першого до останнього, в мові C++ є спеціальний тип ітератора, що дозволяє перебирати елементи у зворотному порядку. Для об'єктів типу `string` це тип `string::reverse_iterator`. В наступному прикладі використовуються функції `rbegin()` та `rend()` для перебору елементів контейнера у зворотному порядку.

```
#include "stdafx.h"
#include <string>
```



```
#include <iostream>
#include <Windows.h>
#include <locale>
using namespace std;

int main()
{
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);
    string речення("Робота з ітераторами");
    // Використання зворотного ітератора
    for (string::reverse_iterator ітератор = речення.rbegin();
        ітератор != речення.rend(); ітератор++)
    {
        cout << *ітератор;
    }
    cout << endl;
    return 0;
}
```

Даний програмний код виводить рядкову змінну типу `string` посимвольно у зворотному порядку за допомогою зворотного ітератора `string::reverse_iterator` та функцій, які з ним використовуються - `rbegin()` та `rend()`. Результатом роботи цієї програми буде наступний рядок:

имаротареті з атобоР

При використанні специфікатора `auto` попередній код у циклі `for` можна дещо спростити:

```
for (auto ітератор = речення.rbegin(); ітератор != речення.rend();
    ітератор++)
{
    cout << *ітератор;
}
```

Специфікатор `auto` автоматично визначає необхідний тип для ітератора у циклі `for`.

З об'єктами типу `string` також можна використовувати серійний цикл `for`, якщо необхідно перебрати змінну посимвольно. В цьому випадку форма запису циклу значно скорочується. В наступному прикладі за допомогою серійного циклу `for` підраховується кількість великих та малих літер «р» у реченні.

```
int кількість(0);
string речення("Робота з ітераторами");
```

```
// Підрахунок літер 'p' та 'P' у реченні
for (auto літера: речення)
{
    if (літера == 'p' || літера == 'P')
    {
        кількість++;
    }
    cout << літера;
}
cout << endl;
cout << "Кількість літер 'p' та 'P' у реченні = " << кількість << endl;
```

Результат роботи даного фрагменту коду наведений нижче:

Робота з ітераторами

Кількість літер 'p' та 'P' у реченні = 3

5.2.4 Операції з ітераторами

Ітератори підтримують лише кілька операцій, які перераховані в табл. 5.2.2.

Таблиця 5.2.2 – Стандартні операції з ітераторами контейнеру

Операція	Пояснення
*iter	Повертає посилання на елемент, позначений ітератором <i>iter</i>
iter->mem	Звернення до значення ітератора <i>iter</i> і обробка члена <i>mem</i> основного елементу. Еквівалентне <i>(*iter).mem</i>
++iter	Інкремент ітератора <i>iter</i> для звернення до наступного елементу контейнеру
--iter	Декремент ітератора <i>iter</i> для звернення до попереднього елементу контейнеру
iter1 == iter2	Порівняння двох ітераторів. Два ітератори рівні, якщо вказують на один і той самий елемент контейнеру або обидва вказують на елемент після останнього того самого контейнеру
iter1 != iter2	Перевірка на нерівність двох ітераторів. Два ітератори нерівні, якщо вказують на різні елементи контейнеру або елементи різних контейнерів

Подібно покажчикам, до значення ітератора можна звернутися (за допомогою оператора звернення до значення ***), щоб отримати елемент, на який він посилається. Можна звертатись до значень лише припустимого ітератора.

Наступний код перетворює рядкові символи в прописні з використанням ітератора

```
string s("some string"); //Пряма ініціалізація рядкової змінної
```



```
//Заголовок оператора for, де задається початкове та кінцеве значення
//змінної циклу (ітератора) та її прирощення
for (string::iterator it = s.begin(); it != s.end(); it++)
{
    //Звернення до значення ітератора (символу рядка) та переведення
    //його у верхній регістр
    *it = toupper(*it);
}
cout << s << endl; //В результаті рядок матиме вигляд SOME STRING
```

В умові оператора `for` записаний логічний оператор `!=` замість оператора `<`. Це пов'язане з тим, що всі контейнери в мові C++ мають ітератори і для них визначені оператори `==` та `!=`. Але лише типи `string` та `vector` мають оператори індексування для яких також визначені оператори `<` та `>`. Щоб не думати про те, для якого контейнеру можна використовувати оператор `<`, програмісти C++ завжди обирають оператор `!=`.

Ітератори використовують оператор інкременту (оператор `++`) для переміщення з одного елементу на наступний. Операція прирощення логічно подібна прирощенню цілого числа. У випадку цілих чисел результатом буде цілочисельне значення збільшене на 1. У випадку ітераторів результатом буде переміщення ітератора на одну позицію вперед.

Ітератори рядків та векторів підтримують додаткові операції, що дозволяють переміщувати ітератори на декілька позицій за раз (наприклад, `iter + n`, `iter - n`, `iter += n`, `iter1 - iter2`). Вони також підтримують всі оператори порівняння. Ці оператори часто називають *арифметичними діями з ітераторами* (`iterator arithmetic`).

Тема 5.3. Масиви

Масив (array) – це структура даних, яка дозволяє зберігати задану кількість *однотипних безіменних* елементів, до яких можна звертатись за позицією в масиві (індексом елементу). Масиви мають фіксований розмір і додавати елементи до масиву не можна.

Якщо завчасно не відома кількість елементів масиву, рекомендується використовувати вектор.

5.3.1 Оголошення та ініціалізація масиву

Масив є складеним типом. *Оператор об'явлення масиву* має форму `a[n]`, де `a` – ім'я масиву, `n` – розмірність масиву. Розмірність задає кількість елементів масиву, вона має бути більше нуля. Кількість елементів – це частина типу масиву, тому вона має бути відома на момент компіляції (до запуску програми на виконання). Відповідно, розмірність має бути *константним виразом*.


```
int n = 10;  
int a[n]; //Помилка!!! Розмірність має бути константним виразом  
string a[10]; //Визначення масиву з 10 пустих рядків
```

Елементи масиву ініціалізуються значеннями за замовчуванням, якщо відсутня явна ініціалізація. Подібно змінним вбудованого типу, ініціалізований за замовчуванням масив вбудованого типу, який визначений у функції, буде містити невизначені значення.

При визначенні масиву необхідно вказати тип його елементів. Подібно вектору, масив містить об'єкти, тому неможливий масив посилань.

Масив підтримує ініціалізацію переліком значень. В цьому випадку розмірність можна опустити.

```
int a2[] = { 1,2,3 }; //Масив з трьох елементів цілочисельного типу
```

Не можна ініціалізувати масив копією іншого масиву, не можна також присвоїти один масив іншому.

5.3.2 Звернення до елементів масиву

Подібно бібліотечним типам `vector` та `string`, для доступу до елементів масиву можна використати *оператор індексування* `[]` (subscript). Як завжди, індекси починаються з 0. Для масиву з 10 елементів використовують індекси від 0 до 9.

При використанні змінної для індексування масиву її визначають типу `size_t`. Тип `size_t` – це машинно-незалежний беззнаковий цілочисельний тип, гарантовано великий для збереження розміру любого об'єкту в пам'яті. Тип `size_t` визначений у заголовку `cstdint`.

Якщо у виразі використати ім'я масиву без вказування індексу його елементу, компілятор замінить його покажчиком на перший елемент масиву. Покажчика на елементи масиву підтримують ті ж операції, що і ітератори. Наприклад можна використати оператор інкременту для переміщення на наступний елемент масиву

```
int arr[] = { 1,2,3,4,5 }; //Масив з п'яти елементів цілочисельного типу  
int *p = arr; //Об'явлення покажчика на тип int та ініціалізація його  
           // першим елементом масиву  
++p; //Тепер покажчик вказує на другий елемент масиву
```

Покажчики подібно ітераторам можна використовувати для перебору елементів масиву. Для цього потрібно отримати покажчики на перший елемент масиву та елемент, наступний за останнім. Для масивів введені дві функції: `begin()` та `end()`. Ці функції працюють аналогічно однойменним функціям для

контейнерів і дозволяють отримати покажчики на перший елемент масиву та наступний за останнім. Однак масиви не є класами, тому не можуть містити функцій-членів. Через це для роботи ці функції в якості аргументу використовують масив (наприклад, `begin(arr)`). Ці функції визначені в заголовці `iterator`.

5.3.3 Масиви масивів

В мові C++ *багатомірних масивів* (multidimensioned array) не існує. Те, що згадують як багатомірний масив в дійсності є *масивом масивів*.

При визначенні масиву, елементи якого є масивами, вказуються дві розмірності: розмірність самого масиву та розмірність його елементів.

```
//Масив з 3 елементів, кожен з яких є масивом з 4 цілих чисел
int arr1[3][4];
//Масив з 10 елементів, кожен з яких є масивом з 20 елементів,
//кожен з яких є масивом з 30 цілих чисел ініціалізованих 0
int arr2[10][20][30] = {0};
```

Подібно любим масивам, елементи багатомірного масиву можна ініціалізувати, навівши у фігурних дужках перелік ініціалізаторів. Багатомірні масиви можуть бути ініціалізовані переліками значень у фігурних дужках для кожного рядку.

```
//Масив з 3 елементів, кожен з яких є масивом з 4 цілих чисел
int arr1[3][4] =
{
    {0, 1, 2, 3},    //ініціалізатори рядку 0
    {4, 5, 6, 7},    //ініціалізатори рядку 1
    {8, 9, 10, 11}   //ініціалізатори рядку 2
};
```

Для доступу до багатомірного масиву можна використовувати індексацію.

```
//Розмірність масиву має бути константним виразом
constexpr size_t rowCnt = 3, colCnt = 4;
//Двомірний масив з 12 неініціалізованих елементів
int arr[rowCnt][colCnt];
for (size_t i = 0; i != rowCnt; i++) // Для кожного рядку
{
    // Для кожного стовпчику в рядку
    for (size_t j = 0; j != colCnt; j++)
    {
        //Присвоїти елементу його індекс як значення
        arr[i][j] = i * colCnt + j;
    }
}
```



```
}
```

5.3.4 Динамічні масиви

У випадку коли розмірність масиву до запуску програми невідома, можна скористатись *динамічним масивом*. Динамічний масив можна задати з використанням покажчиків за допомогою наступної програмної конструкції

```
size_t n; //Об'явлення змінної для розмірності масиву
cout << "Задайте розмірність масиву" << endl;
cin >> n; //Запис розмірності масиву у змінну n
int *arr = new int [n]; //Створення масиву arr для n цілих чисел
for (int *p = arr; p != arr+n; p++) //Запис значень в масив
{
    cout << "Задайте елемент масиву" << endl;
    cin >> *p; //Введення значення елементу масиву
}
delete[] arr; //Звільнення динамічної пам'яті, яка була виділена під масив
```

При створенні динамічного масиву використовують *оператор new[]* – це оператор створення масиву. Він має синтаксис

```
new тип [розмірність]
```

Для звернення до елементів масиву створюється змінна типу покажчик на масив з елементами певного типу

```
int *arr = new int [n];
```

Обов'язковою умовою використання динамічних масивів є *звільнення пам'яті*, яка була виділена для збереження елементів масиву, після завершення роботи з масивом. Ця операція виконується за допомогою *оператора delete* з пустими квадратними дужками та іменем динамічного масиву без вказування індексу чи кількості елементів:

```
delete[] arr;
```

Якщо не застосувати оператор `delete` у зазначеному форматі, масив буде займати оперативну пам'ять комп'ютера навіть після завершення роботи програми. При зазначенні певного індексу у квадратних дужках після імені масиву в операторі `delete` звільниться лише пам'ять, яку займав елемент масиву з вказаним індексом. Оператор `delete` без квадратних дужок використовується для звільнення пам'яті, що була виділена для вказаного після оператора `delete` об'єкту.

Приклад програмної реалізації

Приклад роботи з рядковим типом string та застосуванням ітераторів (Частина 1).

```
// Запам'ятовування окремих слів без пробілів із введеного речення у вектор
// і виведення елементів вектору по 8 слів у рядку

#include "stdafx.h"
#include <iostream>
#include <string>
#include <vector>
#include "windows.h" //Необхідно для використання функцій SetConsoleCP()
                    та SetConsoleOutputCP()

using std::cin;
using std::cout;
using std::endl;
using std::string;
using std::vector;

int main() //використання ітератора для виведення текстових елементів вектора
{
    SetConsoleCP(1251);           //Необхідно для введення в консолі української мови
    SetConsoleOutputCP(1251);     //Необхідно для виведення в консолі української мови

    string word, line, text;
    vector<string> vect1;
    while (getline(cin, line)) //Зчитуємо текст рядками в змінну
        //text поки не буде введено ctrl+z
    {
        text += line + " ";
    }
    cout << "Текст зчитано" << endl;
    cout << text << endl;
    for (string::iterator i = text.begin(); i != text.end(); i++)
        //Розбиваємо текст на слова
    {
        if (isgraph(*i)) //Запам'ятовуємо окреме слово в змінну word
        {
            word += *i;
```

```
    }  
    else  
    {  
        if (word.size() != 0) //Якщо слово закінчилось, додаємо  
                               //його окремим елементом у  
                               //вектор vect1  
        {  
            vect1.push_back(word);  
            word = "";  
        }  
    }  
    cout << endl;  
    cout << "Виводимо елементи вектору по 8 слів у рядку" << endl;  
    unsigned num8 = 0;  
    //Виводимо елементи вектора на екран по 8 в рядку  
    //за допомогою ітератора замість індексації  
    for (vector<string>::const_iterator i = vect1.begin();  
         i != vect1.end(); i++)  
    {  
        cout << *i << " ";  
        num8++;  
        if (num8 == 8)  
        {  
            cout << endl;  
            num8 = 0;  
        }  
    }  
    cout << endl;  
    return 0;  
}
```


Приклад роботи з одномірним динамічним масивом за допомогою вбудованого типу масив та застосуванням покажчиків (Частина 2).

```
// Створення динамічного масиву та запис у нього значень
//ініціалізація вектору масивом

#include "stdafx.h"
#include <iostream>
#include <vector>
#include "windows.h" //Необхідно для використання функцій SetConsoleCP()
                    та SetConsoleOutputCP()

using std::cin;
using std::cout;
using std::endl;
using std::vector;

int main() //Робота з динамічним масивом дійсних чисел та
          //копіювання його значень у вектор
{
    SetConsoleCP(1251);           //Необхідно для введення в консолі
                                  //української мови
    SetConsoleOutputCP(1251);     //Необхідно для виведення в консолі
                                  //української мови

    size_t n, ii=0;
    cout << "Задайте розмірність масиву" << endl;
    cin >> n;
    double *arr = new double [n];
    for (double *arP = arr; arP < arr+n; arP++) //Запис значень у масив
    {
        cout << "Задайте елемент масиву" << endl;
        cin >> *arP;
    }
    cout << endl;
    cout << "Елементи масиву" << endl;
    for (double *arP = arr; arP < arr + n; arP++) //Виведення елементів
                                                    //масиву на екран
    {
        cout << "arr[" << ii << "]= " << *arP << endl;
        ii++;
    }
    cout << endl;
    vector<double> vec(arr, arr+n); //Ініціалізація вектора масивом
```

```
cout << "Елементи вектора" << endl;
ii = 0;
for (vector<double>::iterator iter = vec.begin();
     iter != vec.end(); iter++)
{
    cout << "vec[" << ii << "]= " << *iter << endl;
    ii++;
}
delete[] arr; //Звільнення динамічної пам'яті,
              //яка була виділена під масив

return 0;
}
```

C++

Приклад роботи типу string з кирилицею (Частина 3).

// ConsoleApplication4.cpp: Робота типу string з кирилицею.

```
#include "stdafx.h"
#include <iostream>
#include <string>
#include <vector>
#include "windows.h" //Необхідно для використання
                    //функцій SetConsoleCP() та SetConsoleOutputCP()

using namespace std;

int main()
{
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);
    setlocale(LC_ALL, "Ukrainian"); //Встановлення української локалі
    //для розпізнавання введеної кирилиці
    //string s("приклад тексту");
    //Пряма ініціалізація рядкової змінної
    string s1("приклад тексту"), s2, s3; //Пряма ініціалізація
                                        //рядкової змінної

    s2 = s1;
    cout << "s1 - " << s1 << endl;
    for (string::iterator it = s2.begin(); it != s2.end(); it++)
        //Заголовок оператора for, де задається початкове
        //та кінцеве значення
        //змінної циклу (ітератора) та її прирощення
    {
        *it = toupper(*it);
        //Звернення до значення ітератора (символу рядка)
        //та переведення
        //його у верхній регістр
    }
    cout << "s2 - " << s2 << endl;
    //В результаті рядок матиме вигляд SOME STRING
    if (s1 == s2)
    {
        cout << "Рядки s1 і s2 тотожні" << endl;
    }
    else
    {
        cout << "Рядки s1 і s2 не тотожні" << endl;

        cout << endl << "Зворотна операція:" << endl;
        s3 = s2;
        for (string::iterator it = s3.begin(); it != s3.end(); it++)
        {
```

```
        *it = tolower(*it);  
        //Звернення до значення ітератора (символу рядка)  
        //та переведення його у нижній регістр  
    }  
    cout << "s3 - " << s3 << endl;  
}  
return 0;  
}
```

C++

Завдання до комп'ютерного практикуму №5**Частина 1.**

1. У довільному реченні замінити всі літери «а» на літеру «о» і надрукувати нове речення.
2. Одержати та надрукувати слово, яке утворено з перших букв слів довільного речення.
3. У кожному слові довільного речення замінити перші літери на великі і надрукувати нове речення.
4. Серед слів довільного речення вивести на екран тільки ті слова, що не мають у своєму складі літери «а».
5. У кожному слові довільного речення виділити та надрукувати перші три літери, якщо кількість літер у ньому парна, і перші дві літери – якщо непарна.
6. Організувати введення речення зі знаками пунктуації та вивести речення без знаків пунктуації.
7. Серед слів довільного речення знайти ті, у котрих однакові друга та третя літери.
8. У кожному слові довільного речення поміняти місцями першу та останню літери і надрукувати нове речення.
9. Привести (зменшити) всі слова довільного речення до однакової довжини (довжини найкоротшого слова).
10. Одержати та надрукувати слово, яке утворено з останніх літер слів довільного речення.
11. Організувати зчитування речення в рядкову змінну та замінити пробіли комою і вивести результуючий рядок.
12. Визначити у введеному реченні слова, у які не входить літера «к» та надрукувати їх.
13. Ввести довільне речення, яке містить цілі цифри. Замінити знайдені в реченні цифри знаком оклику.
14. Ввести в рядкову змінну довільне речення. Замінити всі малі літери на великі і вивести результуюче речення.
15. У кожному слові довільного речення поміняти місцями другу та передостанню букви і надрукувати нове речення.
16. Серед слів довільного речення визначити та надрукувати слова, які не починаються літерою «с».
17. Серед слів довільного речення знайти ті, у котрих однакові перша та остання літери.
18. Одержати та надрукувати слово, яке утворено з других літер слів довільного речення.

19. Організувати введення довільного речення зі знаками пунктуації. Вивести результуюче речення, видаливши з нього знаки пунктуації.
20. Організувати введення довільної кількості слів. У кожному слові поміняти першу та останню літеру місцями. Вивести отримані слова.
21. У кожному слові довільного речення замінити останні літери на великі та виведіть отримане речення.
22. Організувати введення довільного речення. Першу літеру кожного слова замінити на порядковий номер слова у реченні.
23. Серед слів довільного речення визначити слова, які мають парну кількість букв, і дописати на початку кожного з цих слів кількість символів, із яких слово складається. Надрукувати речення до та після зміни.
24. Одержати та надрукувати слово, яке утворено з перших букв слів довільного речення та слово, яке утворене з останніх літер .
25. У кожному слові довільного речення замінити літеру «о» на літеру «а», підрахувати загальну кількість замін і надрукувати нове речення.
26. Серед слів довільного речення вивести на екран тільки ті слова, що не мають у своєму складі літери «а».
27. У кожному слові довільного речення виділити та надрукувати перші три літери, якщо кількість літер у ньому парна, і перші дві літери – якщо непарна.
28. Серед слів довільного речення обрати та вивести на екран тільки ті, що починаються або закінчуються на літеру «а».
29. Серед слів довільного речення знайти ті, у котрих однакові перша та третя літери.
30. У кожному слові довільного речення поміняти місцями першу та останню літеру, якщо в слові більше 2 літер, і надрукувати нове речення.

Частина 2.

1. У довільному масиві **Z** із **n** елементів знайти найменший додатний елемент та поміняти його місцями з першим елементом.
2. У довільному масиві **T** із **n** елементів знайти найбільший елемент кратний трьом та поміняти його місцями з останнім елементом.
3. У довільному масиві **S** із **n** елементів знайти суму та кількість від'ємних елементів, кратних чотирьом.
4. У довільному масиві **E** із **n** елементів знайти суму та кількість елементів, значення яких не менше заданої константи ($E_i \geq \alpha$).
5. У довільному масиві **V** із **n** елементів знайти найменший елемент кратний п'яти та поміняти його місцями з першим елементом.
6. У довільному масиві **Z** із **n** елементів знайти найбільший елемент кратний чотирьом та поміняти його місцями з останнім елементом.
7. У довільному масиві **Q** із **n** елементів знайти найбільший від'ємний елемент та поміняти його місцями з першим елементом.
8. У довільному масиві **G** із **n** елементів знайти суму та кількість додатних елементів, кратних п'яти.
9. У довільному масиві **B** із **n** елементів знайти середнє значення B_{cp} , надрукувати елементи, значення яких більше середнього ($B_i > B_{cp}$) та знайти їх суму і кількість.
10. У довільному масиві **B** із **n** елементів знайти найменший елемент кратний двом та поміняти його місцями з першим елементом.
11. У довільному масиві **P** із **n** елементів знайти найбільший від'ємний елемент та поміняти його місцями з останнім елементом.
12. У довільному масиві **W** із **n** елементів знайти суму та кількість від'ємних елементів, кратних трьом.
13. У довільному масиві **H** із **n** елементів знайти найменший елемент кратний трьом та поміняти його місцями з першим елементом.
14. У довільному масиві **U** із **n** елементів знайти суму та кількість від'ємних елементів, кратних п'яти.
15. У довільному масиві **L** із **n** елементів знайти суму та кількість додатних елементів, кратних трьом.
16. У довільному масиві **R** із **n** елементів знайти найбільший елемент кратний двом та поміняти його місцями з останнім елементом.
17. У довільному масиві **R** із **n** елементів знайти найменший елемент кратний чотирьом та поміняти його місцями з першим елементом.
18. У довільному масиві **U** із **n** елементів знайти найбільший та найменший елементи та поміняти їх місцями.

- 19.У довільному масиві **B** із **n** елементів знайти суму та кількість від'ємних елементів, кратних двом.
- 20.У довільному масиві **Y** із **n** елементів знайти найменший елемент кратний п'яти та поміняти його місцями з останнім елементом.
- 21.У довільному масиві **S** із **n** елементів знайти найбільший елемент кратний двом та поміняти його місцями з першим елементом.
- 22.У довільному масиві **Q** із **n** елементів знайти суму та кількість елементів, значення яких знаходяться в межах заданого інтервалу: $\alpha \leq Q_i \leq \beta$.
- 23.У довільному масиві **G** із **n** елементів знайти найменший елемент та поміняти його місцями з останнім елементом.
- 24.У довільному масиві **X** із **n** елементів знайти найменший елемент кратний чотирьом та поміняти його місцями з останнім елементом.
- 25.У довільному масиві **F** із **n** елементів знайти суму та кількість елементів, значення яких не перевищують заданої константи: $F_i \leq \alpha$.
- 26.У довільному масиві **K** із **n** елементів знайти найбільший елемент кратний трьом та поміняти його місцями з першим елементом.
- 27.У довільному масиві **U** із **n** елементів знайти найбільший елемент кратний п'яти та поміняти його місцями з першим елементом.
- 28.У довільному масиві **G** із **n** цілих чисел знайти суму, кількість та середнє значення парних елементів.
- 29.У довільному масиві **Q** із **n** елементів знайти найбільший від'ємний елемент та поміняти його місцями з найменшим додатнім елементом.
- 30.У довільному масиві **H** із **n** елементів знайти найменший елемент кратний трьом та поміняти його місцями з найбільшим елементом кратним трьом.

Контрольні питання

- 1) Для чого потрібний тип `string`, як створити об'єкт даного типу у програмі?
- 2) Як можна ініціалізувати об'єкти типу `string`? Наведіть приклади способів ініціалізації.
- 3) Що таке пряма ініціалізація та ініціалізація копією?
- 4) Перерахуйте основні операції визначені для класу `string`.
- 5) Як можна організувати циклічне введення невизначеної кількості даних?
- 6) Для чого використовується функція `getline()`, у чому полягає її відмінність від оператора `>>`?
- 7) Для чого потрібен допоміжний тип даних `size_type` у класі `string`? Як створити об'єкт даного типу?
- 8) Перерахуйте функції для виконання операцій із символами рядка та їх призначення.
- 9) Що таке оператор індексування? Для чого він потрібен?
- 10) Що таке ітератори? Для чого вони використовуються?
- 11) Для чого потрібні функції-члени `begin()` та `end()` у контейнерах?
- 12) Який тип мають ітератори? Як визначити об'єкт даного типу?
- 13) Перерахуйте та поясніть стандартні операції з ітераторами.
- 14) Як за допомогою ітератора звернутись до значення контейнеру? Чому в циклі `for` при перевірці умови досягнення останнього елементу контейнера використовують оператор `!=` замість оператора `<`?
- 15) Як організувати переміщення по елементам контейнеру за допомогою ітераторів?
- 16) Що таке масив? Як об'явити масив?
- 17) Що таке багатомірний масив? Як можна ініціалізувати масив? Як отримати доступ до елементів масиву за допомогою оператора індексації?
- 18) Що таке динамічний масив? Як об'явити динамічний масив? Як організувати доступ до елементів масиву за допомогою покажчиків?
- 19) Для чого потрібен оператор `delete`? Яка форма його запису?

Комп'ютерний практикум №6

Тип *vector*

Мета: вивчити бібліотечний тип *vector*: *визначення* і *ініціалізація*; додавання елементів; інші операції. Засвоїти використання *оператору індексування* для типу *vector*. Ознайомитись із використанням *ітераторів* для типу *vector*. Опрацювати використання двовірних *векторів*.

Завдання: створити консольний додаток для організації роботи з одномірним та двовірним масивом із використанням бібліотечного типу *vector* для реалізації поставленої задачі згідно отриманого варіанту завдання.

Загальні вимоги.

- 1) Створити консольний додаток з ім'ям виду *PrizvischeKP6*.
- 2) Програмний код модуля має бути чітко структурований.
- 3) Імена об'єктів мають нести сенсові навантаження.
- 4) Програмний код має супроводжуватись коментарями в тексті програми.

Вимоги до виконання.

Частина 1.

- 1) Запрограмувати роботу з одномірним *масивом* за допомогою типу *vector*.
- 2) Звернення до елементів *вектору* організувати з використанням *оператора індексації*.
- 3) Введення вхідних даних організувати в режимі діалогу з користувачем.
- 4) Завдання для обчислень, вхідні дані та результати розрахунків вивести у зовнішній текстовий файл з ім'ям виду *ПрізвищеКП6_з1_OUT*.
- 5) Виведення даних у вихідний файл організувати в режимі додавання в кінець файлу.

Частина 2.

- 1) Запрограмувати роботу з двовірним *динамічним масивом* за допомогою вбудованого типу *vector*.
- 2) Звернення до елементів *вектору* організувати з використанням *ітераторів*.
- 3) Введення вхідних даних організувати з вхідного файлу виду *ПрізвищеКП6_з2_INP*.

- 4) Завдання для обчислень, вхідні дані та результати розрахунків вивести у зовнішній текстовий файл з ім'ям виду *ПрізвищеКП6_з2_OUT*.
- 5) Виведення даних у вихідних файл організувати в режимі додавання в кінець файлу.

Теоретичні відомості

Тема 6.1. Бібліотечний тип `vector`

Вектор (`vector`) – це колекція об'єктів однакового типу, кожному з яких просвоєний цілочисельний індекс, який надає доступ до цього об'єкту. Вектор – це *контейнер* (`container`), оскільки він «містить» інші об'єкти.

Щоб використовувати вектор, необхідно включити відповідний заголовок. Потрібно також включити об'явлення `using`

```
#include <vector>;  
using std::vector;
```

6.1.1 Визначення і ініціалізації векторів

Тип `vector` – це *шаблон класу* (`class template`). Мова C++ підтримує шаблони і класів, і функцій. Шаблони самі по собі не є ні функціями, ні класами. Їх можна вважати інструкцією для компілятора по створенню класів чи функцій. Процес створення компілятором класів чи функцій за шаблоном зветься *створенням екземпляру* (`instantiation`) шаблону. При створенні шаблону необхідно вказати, екземпляр якого класу чи функції має створити компілятор.

Для створення екземпляру шаблону класа слід вказати додаткову інформацію, характер якої залежить від шаблону. Ця інформація завжди задається однаково: в кутових дужках після імені шаблону.

У випадку вектора додатковою інформацією бути тип об'єктів, які він повинен містити:

```
vector<int> vecInt;           //вектор містить об'єкти типу int  
vector<string> vecStr;       //вектор містить об'єкти типу string  
vector<MyStruct> vecMyStr;    //вектор містить об'єкти типу MyStruct,  
                             //створеного користувачем
```

Можна визначити вектори для зберігання об'єктів практично любого типу. Оскільки посилання не є об'єктами, не може бути вектору посилань. Також може бути вектор, елементами якого є інший вектор.

```
//Вектор, елементами якого є вектори, що містять текстові рядки  
vector<vector<string>> vecVecStr;
```


Подібно будь-якому типу класа, шаблон `vector` контролює спосіб визначення і ініціалізації векторів. Найбільш розповсюджені способи визначення векторів наведені в табл. 6.1.1.

Таблиця 6.1.1 – Способи ініціалізації об'єкту класу `vector`

Ініціалізація	Роз'яснення
<code>vector<T> v1</code>	Вектор, що містить об'єкти типу <code>T</code> . Стандартний конструктор, <code>v1</code> пустий
<code>vector<T> v2(v1)</code>	Вектор <code>v2</code> - копія всіх елементів вектору <code>v1</code>
<code>vector<T> v2 = v1</code>	Еквівалент <code>v2(v1)</code> , вектор <code>v2</code> - копія елементів вектору <code>v1</code>
<code>vector<T> v3(n, val)</code>	Вектор <code>v3</code> містить <code>n</code> елементів зі значенням <code>val</code>
<code>vector<T> v4(n)</code>	Вектор <code>v4</code> містить <code>n</code> екземплярів об'єкту типу <code>T</code> , ініціалізованих значенням за замовчуванням
<code>vector<T> v5{a,b,c ...}</code>	Вектор <code>v5</code> містить стільки елементів, скільки вказано у фігурних дужках. Елементи вектору ініціалізуються вказаними у фігурних дужках значеннями (ініціалізація переліком)
<code>vector<T> v5 = {a,b,c ...}</code>	Еквівалент <code>v5{a,b,c ...}</code> . Пряма ініціалізація

Ініціалізація вектор за замовчуванням дозволяє створити пустий вектор певного типу.

```
//Ініціалізація за замовчуванням, у вектора sVec немає елементів
vector<string> sVec;
```

Вектор використовується в мові C++ замість масивів. Шаблон `vector` створений на основі типу масив з додаванням у нього необхідних властивостей. На відміну від масивів, у вектор *можна додавати елементи*. Тому створення пустого вектору певного типу є звичайною операцією з метою подальшого наповнення вектору елементами вказаного типу.

При визначенні вектору для його елементів можна також вказати початкові значення. Наприклад, можна скопіювати елементи з іншого вектора (на відміну від масивів, які копіювати не можна). При копіюванні векторів кожен елемент нового вектору буде копією відповідного елементу вихідного вектору. Обидва вектори повинні мати однаковий тип:

```
vector<int> vecInt(10, 1); //вектор містить десять одиниць
vector<int> iVec1 = vecInt; //вектор iVec1 містить десять одиниць
vector<int> iVec2(iVec1); //вектор iVec2 містить десять одиниць
```

Вектори підтримують *ініціалізацію переліком*, коли значення задаються у фігурних дужках після імені вектору:

```
//Вектор articles міститиме три рядкових елементи a, an та the
```



```
vector<string> articles{"a","an","the"};
```

При ініціалізації вектора значення можна пропустити, а вказати тільки кількість елементів. В такому випадку відбудеться *ініціалізація значення* (value initialization), тобто бібліотека створить ініціалізатор елементу самаю. Кількість елементів можна вказувати тільки за допомогою прямої ініціалізації – у круглих дужках після імені вектору:

```
vector<int> iVec(10); //вектор з 10 нулів  
vector<string> iVec(10); //вектор з 10 пустих рядків
```

Використання круглих та фігурних дужок дає різні результати:

```
vector<int> vec1(10); //вектор містить 10 елементів зі значенням 0  
vector<int> vec2{ 10 }; //вектор містить один елемент зі значенням 10  
vector<int> vec3(10,1); //вектор містить 10 елементів зі значенням 1  
vector<int> vec4{ 10, 1 }; //вектор містить містить два елементи  
                        //зі значенням 10 та 1
```

6.1.2 Функція push_back()

Зазвичай при створенні вектора невідома кількість його елементів та їх значення. Для додавання елементів у вектор використовують функцію `push_back()`, яка додає новий елемент в кінець вектора.

```
vector<int> v1; //пустий вектор  
for (size_t i = 0; i != 10; i++)  
{  
    v1.push_back(i + 1); //додавання елементу в кінець вектора зі  
                        //значенням i + 1  
    cout << v1[i] << endl; //виведення значення вектора на екран  
}
```

Результатом роботи даного приклада буде виведення на екран у стовпчик значень створеного вектора, якими будуть числа від 1 до 10.

6.1.3 Операції з векторами

Окрім функції `push_back()`, шаблон `vector` надає ще декілька операцій, більшість з яких подібна операціям класу `string`. Найбільш важливі з них наведені в табл. 6.1.2.

Таблиця 6.1.2 – Операції з векторами

Операція	Роз'яснення
<code>v.push_back(t)</code>	Додає елемент зі значенням <code>t</code> в кінець вектора <code>v</code>
<code>v.empty()</code>	Повертає значення <code>true</code> , якщо вектор <code>v</code> пустий. Інакше повертає значення <code>false</code>
<code>v.size()</code>	Повертає кількість елементів вектору <code>v</code>
<code>v[n]</code>	Повертає посилання на елемент у позиції <code>n</code> вектора <code>v</code> ; позиції відраховуються від 0
<code>v1 = {a,b,c ...}</code>	Замінює елементи вектору <code>v1</code> копією елементів із розділеного комами переліку
<code>v1 = v2</code>	Замінює елементи вектору <code>v1</code> копією елементів вектору <code>v2</code>
<code>v1 == v2</code>	Вектори <code>v1</code> та <code>v2</code> рівні, якщо кількість елементів рівна і вони містять однакові елементи на тих самих позиціях
<code>v1 != v2</code>	Вектори <code>v1</code> та <code>v2</code> не рівні, якщо хоча б один елемент відрізняється чи різна кількість елементів у векторах
<code><, <=, >, >=</code>	Мають звичне значення і покладаються на алфавітний порядок символів

Доступ до елементів вектору здійснюється так само, як і до символів у рядку: за їх позицією у векторі.

6.1.4 Функції `empty()` та `size()`

Функції-члени `empty()` та `size()` працюють аналогічно, як і в класі `string`. Функція-член `size()` повертає значення типу `size_type`, яке визначене відповідним типом шаблону `vector`, наприклад `vector<string>::size_type`.

За допомогою *оператора індексування* можна вибрати вказаний елемент. Подібно рядкам, індексування вектора починається з 0; індекс має тип `size_type` відповідного типу; якщо вектор не константний, то у повернутий оператором індексування елемент можна здійснити запис; можна розрахувати індекс і безпосередньо звернутись до елементу в даній позиції. За допомогою оператора індексування *неможливо додати елементи* у вектор, для цього використовується лише функція `push_back()`. Оператор індексування дозволяє звертатися (в тому числі і змінювати) лише до вже існуючих елементів вектору.

6.1.5 Використання ітераторів з векторами

Тип `vector` є контейнером і підтримує роботу з *ітераторами* аналогічно типу `string`. Вектор використовує функції-члени `begin()` та `end()` для звернення до першого та наступного за останнім елементу вектора.

```
vector<float> v{ 0.1, 0.2, 0.3, 0.4, 0.5 }
//Використання ітератора для звернення до елементів вектору
```



```
for (vector<float>::iterator iter = v.begin(); iter != v.end(); iter++)  
{  
    //Виведення на екран елементів вектору в рядок  
    cout << *iter << " ";  
}
```

Оскільки вектор може містити інші вектори, то, наприклад, для збереження елементів матриці використовують вектор векторів. Так для збереження матриці з цілих чисел необхідно створити наступний вектор:

```
vector<vector<int>> vec1;
```

Для спрощення формату запису складних типів використовують псевдоніми типів. *Псевдонім типу* (type alias) – це ім'я, яке є синонімом імені типу. Псевдонім типу дозволяє спростити складні визначення типів, полегшуючи їх використання. Псевдоніми типу дозволяють також підкреслити мету використання типу. Визначити псевдонім типу можна одним з двох способів. Традиційно для цього використовують ключове слово `typedef`:

```
typedef double wages; //wages – синонім для double  
typedef vector<vector<float>> matrix; //matrix – синонім для двовимірного  
//вектора (вектора векторів) дійсних чисел  
wages w1; //еквівалент об'явлення double w1;  
matrix m1; //еквівалент об'явлення vector<vector<float>> m1;
```

Іншим способом створення псевдоніму типу є *об'явлення псевдоніму* (alias declaration) за допомогою ключового слова `using` та знаку `=`.

```
using wages = double;  
wages w1;
```

Щоб звернутись до значення вектору можна скористатись ітератором з *оператором звернення до значення* `*`, наприклад `*iter`. Якщо ми створили вектор векторів, то звернення до значення вектора векторів поверне об'єкт, яким буде вектор.

При зверненні до значення ітератора отримуємо об'єкт, на який вказує ітератор. Якщо цей об'єкт має тип класу, то може знадобитися доступ до члену отриманого об'єкту. Наприклад, якщо є вектор рядків, то може знадобитися дізнатись, чи не пустий певний елемент. З урахуванням того, що `it` – це ітератор даного вектору, можна наступним чином перевірити чи не пустий рядок, на який вказує ітератор:

```
(*it).empty()
```


Круглі дужки у виразі необхідні, бо вони вимагають застосувати *оператор звернення до значення* (*) до ітератора *it*, а до повернутого ітератором значення застосувати *точковий оператор* (.) (оператор звернення до члену класу). Якщо б не було круглих дужок, точковий оператор відносився б до ітератору *it*, а не до повернутого ним об'єкту.

Для спрощення таких виразів, мова пропонує *оператор стрілки* (->) (arrow operator). Оператор стрілки об'єднує оператори звернення до значення і доступ до члену. Таким чином, вираз `(*it).empty()` можна переписати у спрощеному вигляді:

```
it->empty()
```

Використання векторів разом з ітераторами надає більш гнучкий інструмент для роботи з наборами однотипних елементів, ніж масиви. Всі вектори, на відміну від масивів, є динамічними, дозволяють додавати значення, копіювати та присвоювати вектори, не вимагають програмування звільнення пам'яті.

Тема 6.2. Генератор випадкових чисел

Програми часто потребують джерело випадкових чисел. До стандарту C++ 11 для цього використовувалась проста бібліотечна функція мови C на ім'я `rand()`. Дана функція створює псевдовипадкові цілі числа, які рівномірно розподілені в діапазоні від 0 до залежного від системи максимального значення, котре не менше 32767. Але більшості програм потрібні випадкові числа в зовсім іншому діапазоні. Деяким додаткам потрібні випадкові числа з плаваючою комою, або числа з неоднорідним розподілом.

Ці проблеми вирішує бібліотека випадкових чисел, яка визначена у заголовку `random`. Дана бібліотека включає набір взаємодіючих класів: класів *процесора випадкових чисел* (random-number engine) та класів *розподілу випадкового числа* (random-number distribution).

Процесор створює послідовність беззнакових випадкових чисел, а розподіл використовує процесор для створення випадкових чисел визначеного типу в заданому діапазоні, який розподілений згідно вказаного імовірнісного розподілу.

6.2.1 Генерування цілих випадкових чисел в заданому діапазоні

В C++ використовують клас `default_random_engine` для генерування простих випадкових чисел, але ці числа зазвичай лежать у відмінному від потрібного діапазону. Для отримання числа в певному діапазоні, використовують об'єкт типу розподілу.


```
#include <random> //бібліотека генерування випадкових чисел
int main()
{
    // Створює випадкове беззнакове ціле число
    std::default_random_engine e;
    // Повертає однорідно розподілене ціле число в заданому діапазоні
    std::uniform_int_distribution<int> u(-5, 5);
    // Цикл виводить 10 чисел з заданого діапазону
    for (int i = 0; i < 10; i++)
    {
        // Випадкове ціле число з заданого діапазону
        std::cout << u(e) << " ";
    }
    return 0;
}
```

Результатом роботи програми буде наступний рядок чисел

1 -2 5 4 -1 2 -5 1 0 -1

Тут `u` визначається як об'єкт типу `uniform_int_distribution<int>`. Цей тип створює однорідно розподілені значення. При визначенні об'єкту даного типу можна задати мінімум та максимум необхідних значень. Визначення `u(-5, 5)` вказує, що необхідні числа в діапазоні від -5 до 5 включно. Розподіл випадкового числа використовує включений діапазон, який дозволяє отримати будь-яке можливе цілочисельне значення в ньому.

6.2.2 Генерування випадкових чисел з плаваючою комою в заданому діапазоні

Для отримання набору випадкових чисел з плаваючою комою необхідно визначити об'єкт типу `uniform_real_distribution<double>`. Подібно до типу `uniform_int_distribution<int>`, тут також можна задати мінімальне та максимальне значення при визначенні об'єкту

```
#include <random> // Бібліотека генерування випадкових чисел
int main()
{
    // Створює випадкове беззнакове ціле число
    std::default_random_engine e;
    // Повертає однорідно розподілене дійсне число в заданому діапазоні
    std::uniform_real_distribution<double> u(-5, 5);
    // Цикл виводить 5 чисел з заданого діапазону
```



```
for (int i = 0; i < 5; i++)
{
    // Випадкове ціле число з заданого діапазону
    std::cout << u(e) << " ";
}
return 0;
}
```

Результатом роботи програми буде наступний рядок чисел

-3.64523 3.35009 4.68868 -2.78966 -1.91833

6.2.3 Використання функції `time()` для генерування випадкових чисел

В обох попередніх прикладах при кожному запуску програми буде генеруватись та ж сама послідовність випадкових чисел. Для отримання кожного разу нового набору випадкових чисел можна скористатись системною функцією `time()`, яка визначена у заголовку `ctime`. Функція `time()` повертає кількість секунд, починаючи з заданої епохи. Дана функція отримує один параметр, який є покажчиком на структуру для запису часу. Якщо цей покажчик нульовий, функція лише повертає час

```
default_random_engine e(time(0)); //майже випадкове початкове число
```

З використанням функції `time()` попередній приклад набуде наступного вигляду

```
#include <random> // Бібліотека генерування випадкових чисел
#include <ctime> // Бібліотека, яка містить функцію time()
int main()
{
    // Створює випадкове беззнакове ціле число
    std::default_random_engine e(time(0));
    // Повертає однорідно розподілене дійсне число в заданому діапазоні
    std::uniform_real_distribution<double> u(-5, 5);
    // Цикл виводить 5 чисел з заданого діапазону
    for (int i = 0; i < 5; i++)
    {
        // Випадкове ціле число з заданого діапазону
        std::cout << u << " ";
    }
    return 0;
}
```


Результатом роботи програми буде наступний рядок чисел

0.912443 -4.218 -3.05783 4.38754 4.08929

При новому запуску програми набір чисел буде інший

1.49806 -0.775143 -4.46224 -4.58333 0.434274

C++

Приклад програмної реалізації

Приклад роботи з однимірним масивом за допомогою типу vector та застосуванням індексації (Частина 1).

```
// Розрахувати суми 1-го і останнього, 2-го і передостаннього ...
//елементів вектору

#include "stdafx.h"
#include <iostream>;
#include <vector>;
#include "windows.h" //Необхідно для використання функцій SetConsoleCP()
//та SetConsoleOutputCP()

using std::cin;
using std::cout;
using std::endl;
using std::vector;

int main()
{
    SetConsoleCP(1251);           //Необхідно для введення в консолі
                                  української мови
    SetConsoleOutputCP(1251);     //Необхідно для виведення в консолі
                                  української мови

    vector<int> intV;
    int num;
    while (cin >> num) //Задати вектор із довільних цілих чисел
    {
        intV.push_back(num);
    }

    for (vector<int>::size_type i = 0; i != intV.size(); i++)
        //Просумувати 1-й і останній, 2-й та
        //передостанній і т.д. елементи вектору
    {
        if (i<(intV.size() - i - 1)) //шукаємо середину вектора
        {
            cout << "Сума двох протилежних елементів вектору "
                 << intV[i] << " та "
                 << intV[intV.size() - i - 1]
                 << " = " << intV[i] + intV[intV.size() - i - 1]
                 << endl;
        }
    }
}
```



```
}  
return 0;  
}
```

C++

Приклад роботи з двовірним динамічним масивом за допомогою типу `vector` та застосуванням ітераторів (Частина 2).

```
// Створення двовірного вектору та запис у нього значень
//виведення елементів двовірного вектору на екран

#include "stdafx.h"
#include <iostream>
#include <vector>
#include "windows.h" //Необхідно для використання функцій SetConsoleCP()
                    та SetConsoleOutputCP()

using std::cin;
using std::cout;
using std::endl;
using std::vector;

int main() //Ілюстрація використання двовірних векторів для задавання
          //динамічних матриць (замість динамічних двовірних масивів)
{
    SetConsoleCP(1251);          //Необхідно для введення в консолі
                                //української мови
    SetConsoleOutputCP(1251);    //Необхідно для виведення в консолі
                                //української мови

    size_t n, m;
    cout << "Задайте розмірність матриці n - рядки" << endl;
    cin >> n;
    cout << "Задайте розмірність матриці m - стовпчики" << endl;
    cin >> m;
    cout << endl;
    typedef vector<vector<double>> StrMat; //Створення нового типу,
                                           //вектору що складається з векторів
                                           //дійсних чисел (зовнішній вектор)
    typedef vector<double> StrVec; //Створення нового типу - вектору
                                   //з дійсних чисел (внутрішній вектор)
    StrMat Matrix; //Створення нової змінної типу зовнішнього вектору
    StrVec Vect;   //Створення нової змінної типу внутрішнього вектору
    for (size_t i = 0; i != n; i++) //Задавання елементів зовнішнього
                                   //вектору
    {
        for (size_t j = 0; j != m; j++) //Задавання елементів
                                        //внутрішнього вектору
        {
```



```
        //Залежність для розрахунку елемента внутрішнього
        // вектору і додавання його в кінець
        Vect.push_back((cos(j * m + i * n)+ m) / (sin(j * n
+ i * m) + n));
    }
    Matrix.push_back(Vect); //Додавання в зовнішній вектор
    //елементу (внутрішній вектор) у кінець
    Vect.clear(); //Очищення внутрішнього вектору для
    //додавання нових елементів
}
cout << "Результуюча матриця" << endl;
//Використання ітератора для зовнішнього вектору
for (StrMat::iterator MatIter = Matrix.begin();
    MatIter != Matrix.end(); MatIter++)
{
    //Використання ітератора для внутрішнього вектору
    for (StrVec::iterator VecIter = MatIter->begin();
        VecIter != MatIter->end(); VecIter++)
    {
        cout << *VecIter << " "; //Виведення на екран елементів
        //вектору в рядок
    }
    cout << endl; //Перехід на новий рядок для виведення
    //наступного рядку матриці
}
cout << endl;
return 0;
}
```

Пошук сум елементів в парних та непарних рядках матриці, а також сум елементів з непарними та парними індексами в рядку (Частина 3).

```
// Matrix.cpp: Пошук сум елементів в парних та непарних рядках матриці,  
//а також сум елементів з непарними та парними індексами в рядку  
  
#include "stdafx.h"  
#include <iostream>  
#include <random> // Бібліотека генерування випадкових чисел  
#include <ctime> // Бібліотека, яка містить функцію time()  
#include <vector>  
#include <iomanip> // Бібліотека для маніпулятора setw  
#include <locale> // Бібліотека для функції setlocale  
#include <Windows.h>  
  
using namespace std;  
  
int main()  
{  
    SetConsoleCP(1251); // Встановлення таблиці кодування  
    SetConsoleOutputCP(1251); // Встановлення таблиці кодування  
    size_t n, m;  
    cout << "Задайте розмірність матриці n- рядки" << endl;  
    cin >> n;  
    cout << "Задайте розмірність матриці m-стовпчики" << endl;  
    cin >> m;  
    cout << endl;  
    // Створення нового типу, вектору  
    // що складається з векторів Цілих чисел (зовнішній вектор)  
    typedef vector<vector<int>> StrMat;  
    typedef vector<int> StrVec; // Створення нового типу - вектору  
                                // з Цілих чисел (внутрішній вектор)  
    StrMat Matrix; // Створення нової змінної типу зовнішнього вектору  
    StrVec Vect; // Створення нової змінної типу внутрішнього вектору  
    cout << "Друк матриці з використанням індексації" << endl;  
    for (size_t i = 0; i != n; i++) // Задавання елементів зовнішнього  
                                    //вектору  
    {  
        for (size_t j = 0; j != m; j++) // Задавання елементів  
                                        //внутрішнього вектору  
        {  
            // Додавання нового елементу в кінець внутрішнього вектору  
            // (задавання елементів внутрішнього вектора):  
            Vect.push_back(rand() % 100 - rand() % 50);  
            cout << setw(4) << Vect[j] << " ";  
        }  
        cout << endl;  
        Matrix.push_back(Vect); // Додавання в зовнішній вектор  
                                // елементу (внутрішній вектор) в кінець
```



```

        Vect.clear();    // Очищення внутрішнього вектору для
                        // додавання нових елементів
    }
    int i = 0;
    cout << endl << "Друк матриці з використанням ітераторів";
    for (StrMat::iterator MatIter = Matrix.begin();
        MatIter != Matrix.end(); MatIter++)    // Цикл для зовнішнього
                                                //вектора
    {
        // Друк порядкового номеру рядка
        cout << endl << "Рядок №" << i << " ";
        //Цикл для внутрішнього вектора:
        for (StrVec::iterator VecIter = MatIter->begin();
            VecIter != MatIter->end(); VecIter++)
        {
            // Друк елементів внутрішнього вектора (рядків матриці)
            //з використанням ітератора
            cout << setw(4) << *VecIter;
        }
        i++;
    }
    cout << endl << endl;
    cout << "Непарні рядки матриці" << endl;
    i = 0;
    for (size_t i = 0; i < n; i += 2)    // Обираємо лише непарні рядки
    {
        int sum = 0;
        // Друкуємо порядковий номер рядка
        cout << "Рядок №" << i << " ";
        for (size_t j = 0; j < m; j++)
        {
            cout << setw(4) << Matrix[i][j];
            // Рахуємо суму елементів непарних рядків матриці
            sum += Matrix[i][j];
        }
        // Друкуємо суму елементів непарних рядків матриці:
        cout << " Сума елементів рядка = " << sum << endl;
    }
    cout << endl;
    cout << "Суми елементів рядків з непарними індексами" << endl;
    i = 0;
    for (size_t i = 0; i < n; i++)
    {
        int sum1 = 0;
        // Друкуємо порядковий номер рядка
        cout << "Рядок №" << i << " ";
        // Рахуємо суму елементів з непарними індексами
        //в рядку матриці:
        for (size_t j = 0; j < m; j += 2)

```

```
{
    cout << setw(4) << Matrix[i][j];
    sum1 += Matrix[i][j];
}
cout << " Сума елементів = " << setw(4) << sum1 << endl;
}
cout << endl;
cout << "Суми елементів рядків з парними індексами" << endl;
i = 0;
for (size_t i = 0; i < n; i++)
{
    int sum2 = 0;
    // Друкуємо порядковий номер рядка
    cout << "Рядок №" << i << " ";
    // Рахуємо суму елементів з парними індексами в рядку матриці:
    for (size_t j = 1; j < m; j += 2)
    {
        cout << setw(4) << Matrix[i][j];
        sum2 += Matrix[i][j];
    }
    cout << " Сума елементів = " << setw(4) << sum2 << endl;
}
cout << endl;
return 0;
}
```


Сортування вектора векторів (матриці) за збільшенням, зменшенням елементів матриці у рядках та пошук мінімального елементу у рядку (Частина 4).

// ConsoleApplicationVector2.cpp: Програма сортування вектора векторів (матриці) за збільшенням, зменшенням елементів матриці у рядках та пошук мінімального елементу у рядку
//

```
#include "stdafx.h"
#include <iostream>
#include <vector>
#include <algorithm> //Бібліотека для функції sort
#include <functional> // Бібліотека для функції greater<int>( )
#include <iomanip> ///Бібліотека для маніпулятора форматування setw
#include <locale> //Бібліотека для зміни таблиці кодування символів
                //при виведенні на екран функцією setlocale

#include <Windows.h>
using namespace std;

//Ілюстрація використання двовірних векторів для задавання динамічних
//матриць (замість динамічних двовірних масивів)
int main()
{
    // установка кодової сторінки win-ср 1251 в потік введення
    SetConsoleCP(1251);
    // установка кодової сторінки win-ср 1251 в потік виведення
    SetConsoleOutputCP(1251);
    size_t n, m;
    cout << "Задайте розмірність матриці n - рядки" << endl;
    cin >> n;
    cout << "Задайте розмірність матриці m - стовпчики" << endl;
    cin >> m;
    cout << endl;
    //Створення нового типу, вектору що складається з векторів
    //дійсних чисел (зовнішній вектор):
    typedef vector<vector<int>> StrMat;
    typedef vector<int> StrVec; //Створення нового типу - вектору
                                //з дійсних чисел (внутрішній вектор)
    StrMat Matrix; //Створення нової змінної типу зовнішнього вектору
    StrVec Vect;   //Створення нової змінної типу внутрішнього вектору
    for (size_t i = 0; i != n; i++) //Задавання елементів
                                    //зовнішнього вектору
    {
        for (size_t j = 0; j != m; j++) //Задавання елементів
                                        //внутрішнього вектору
        {
            //Залежність для розрахунку елементу внутрішнього
            //вектору і додавання його в кінець:

```



```
Vect.push_back(rand() % 25 - rand() % 10);
}
Matrix.push_back(Vect); //Додавання в зовнішній вектор
                           //елементу (внутрішній вектор) в кінець
Vect.clear(); //Очищення внутрішнього вектору для додавання
               //нових елементів
}
cout << "Сформована матриця" << endl;
//Використання ітератора для зовнішнього вектору:
for (StrMat::iterator MatIter = Matrix.begin();
     MatIter != Matrix.end(); MatIter++)
{
    int min = 1000;
    //Використання ітератора для внутрішнього вектору:
    for (StrVec::iterator VecIter = MatIter->begin();
         VecIter != MatIter->end(); VecIter++)
    {
        //Виведення на екран елементів вектору
        //в рядок з вирівнюванням:
        cout << setw(4) << *VecIter;
        if (*VecIter < min)
        {
            min = *VecIter;
        }
    }
    //Перехід на новий рядок для виведення наступного
    //ряду матриці:
    cout << " Мінімальний елемент в рядку = " << min << endl;
}
cout << endl;
//Виведення на екран елементів вектору:
cout << "Матриця відсортована за збільшенням елементів у рядках"
    << endl;
//Використання ітератора для зовнішнього вектору:
for (StrMat::iterator MatIter = Matrix.begin();
     MatIter != Matrix.end(); MatIter++)
{
    // Сортування внутрішніх векторів за збільшенням
    // (сортування матриці за збільшенням
    //елементів у рядках):
    sort(MatIter->begin(), MatIter->end());
    //Використання ітератора для внутрішнього вектору:
    for (StrVec::iterator VecIter = MatIter->begin();
         VecIter != MatIter->end(); VecIter++)
    {
        //Виведення на екран елементів вектору в рядок
        //з вирівнюванням:
        cout << setw(4) << *VecIter;
    }
}
```



```
        //Перехід на новий рядок для виведення наступного
        //ряду матриці
        cout << endl;
    }
    cout << endl;
    //Виведення на екран елементів вектору:
    cout << "Матриця відсортована за зменшенням елементів у рядках"
        << endl;
    //Використання ітератора для зовнішнього вектору:
    for (StrMat::iterator MatIter = Matrix.begin();
        MatIter != Matrix.end(); MatIter++)
    {
        // Сотрування внутрішніх векторів за зменшенням
        // (сортуювання матриці за зменшенням елементів у рядках):
        sort(MatIter->begin(), MatIter->end(), greater<int>());
        //Використання ітератора для внутрішнього вектору:
        for (StrVec::iterator VecIter = MatIter->begin();
            VecIter != MatIter->end(); VecIter++)
        {
            //Виведення на екран елементів вектору в рядок
            //з вирівнюванням:
            cout << setw(4) << *VecIter;
        }
        //Перехід на новий рядок для виведення наступного
        //ряду матриці
        cout << endl;
    }
    cout << endl;
    system("pause");
    return 0;
}
```


Завдання до комп'ютерного практикуму №6

Частина 1.

1. У довільному масиві B із n дійсних елементів знайти середнє значення B_{cp} , надрукувати елементи, значення яких більше середнього ($B_i > B_{cp}$) та знайти їх суму, кількість та середнє значення.
2. У довільному масиві G із n дійсних елементів знайти суму та кількість додатних елементів, ціла частина яких кратна п'яти.
3. У довільному масиві Q із n дійсних додатніх та від'ємних елементів знайти найбільший від'ємний елемент та поміняти його місцями з першим додатнім елементом.
4. У довільному масиві Z із n елементів знайти найбільший елемент кратний чотирьом та поміняти його місцями з останнім елементом кратним двом.
5. У довільному масиві V із n дійсних елементів знайти найменший елемент, ціла частина якого кратна п'яти, та поміняти його місцями з першим елементом.
6. У довільному масиві E із n дійсних елементів знайти суму та кількість елементів, значення яких не менше заданої константи ($E_i \geq \alpha$).
7. У довільному масиві S із n дійсних елементів знайти суму та кількість від'ємних елементів, ціла частина яких кратна трьом.
8. У довільному масиві U із n елементів знайти найбільший елемент кратний п'яти та поміняти його місцями з першим від'ємним елементом.
9. У довільному масиві K із n елементів знайти найбільший елемент кратний трьом та поміняти його місцями з першим не кратним трьом елементом.
10. У довільному масиві F із n дійсних додатніх та від'ємних елементів знайти суму та кількість елементів, значення яких за модулем не перевищують заданої константи: $F_i \leq \alpha$.
11. У довільному масиві X із n додатніх та від'ємних елементів знайти найменший елемент кратний чотирьом та поміняти його місцями з останнім від'ємним елементом.
12. У довільному масиві G із n дійсних додатніх та від'ємних елементів знайти найменший за модулем елемент та поміняти його місцями з останнім додатнім елементом.
13. У довільному масиві X із n дійсних елементів знайти суму та кількість від'ємних елементів, а також суму та кількість елементів, значення яких більше ніж середнє значення елементів масиву.
14. У довільному масиві Q із n дійсних додатніх та від'ємних елементів знайти суму та кількість елементів, значення за модулем яких знаходяться в межах заданого інтервалу: $\alpha \leq Q_i \leq \beta$.

15. У довільному масиві **G** із **n** дійсних від'ємних та додатніх елементів знайти найменший елемент та поміняти його місцями з останнім додатнім елементом масиву.
16. У довільному масиві **S** із **n** елементів знайти найбільший елемент кратний двом та поміняти його місцями з найменшим елементом.
17. У довільному масиві **Y** із **n** додатніх та від'ємних елементів знайти найменший елемент кратний п'яти та поміняти його місцями з останнім від'ємним елементом.
18. У довільному масиві **T** із **n** елементів знайти найбільший елемент кратний трьом та поміняти його місцями з останнім елементом кратним двом.
19. У довільному масиві **B** із **n** елементів знайти суму та кількість від'ємних елементів, кратних двом, а також суму та кількість додатніх елементів, кратних двом.
20. У довільному масиві **U** із **n** елементів знайти найбільший від'ємний та найменший додатній (крім нуля) елементи та поміняти їх місцями.
21. У довільному масиві **R** із **n** додатніх та від'ємних елементів знайти найбільший від'ємний елемент кратний чотирьом та поміняти його місцями з першим від'ємним елементом.
22. У довільному масиві **R** із **n** елементів знайти найбільший елемент кратний двом та поміняти його місцями з останнім кратним трьом елементом
23. У довільному масиві **T** із **n** дійсних елементів знайти найбільший елемент, ціла частина якого кратна трьом та поміняти його місцями з останнім елементом.
24. У довільному масиві **L** із **n** дійсних елементів знайти суму та кількість додатніх елементів, цілі частини яких кратні двом.
25. У довільному масиві **U** із **n** елементів знайти суму та кількість елементів, кратних двом та менших за середнє значення елементів масиву.
26. У довільному масиві **H** із **n** елементів знайти найменший елемент кратний трьом та поміняти його місцями з середнім елементом масиву, якщо він є, інакше - з елементом $n/2$.
27. У довільному масиві **W** із **n** елементів знайти суму та кількість від'ємних елементів, кратних трьом, а також суму та кількість додатніх елементів.
28. У довільному масиві **P** із **n** дійсних додатніх та від'ємних елементів знайти найбільший від'ємний елемент та поміняти його місцями з останнім додатнім елементом.
29. У довільному масиві **B** із **n** елементів знайти найменший елемент кратний двом та поміняти його місцями з першим елементом кратним трьом.
30. У довільному масиві **Z** із **n** дійсних додатніх та від'ємних елементів знайти найменший додатній елемент та поміняти його місцями з першим від'ємним елементом.

Частина 2.

1. Задано довільну цілочисельну матрицю $S(m, n)$. Діапазон значень елементів матриці $[-50, 50]$. В 1, 4, 7 і т.д. стовпцях матриці (окремо) визначити мінімальні за модулем елементи та їх координати. Матрицю S , значення та координати мінімальних елементів надрукувати.
2. Задано довільну цілочисельну матрицю $U(n, n)$. Діапазон значень елементів матриці $[-350, 250]$. Для елементів матриці U , розташованих на її головній діагоналі знайти всі додатні елементи кратні трьом, їх суму, кількість та середнє значення. Матрицю U , отримані суму, кількість та середнє значення надрукувати.
3. Задано довільну цілочисельну матрицю $B(m, n)$. Діапазон значень елементів матриці $[-200, 200]$. В 1, 4, 7 і т.д. стовпцях матриці (окремо) визначити всі додатні елементи кратні трьом, їх суми, кількості та середні значення. Матрицю B , отримані суми, кількості та середні значення надрукувати.
4. Задано довільну цілочисельну матрицю $E(m, n)$. Діапазон значень елементів матриці $[-250, 50]$. У кожному рядку матриці окремо знайти суми, кількості та середні значення від'ємних елементів кратних трьом. Матрицю, отримані суми, кількості та середні значення надрукувати.
5. Задано довільну цілочисельну матрицю $F(m, n)$. Діапазон значень елементів матриці $[-200, 300]$. В 2, 5, 8 і т.д. рядках матриці (окремо) знайти суми, кількості та середні значення від'ємних елементів кратних трьом. Матрицю F , отримані суми, кількості та середні значення надрукувати.
6. Задано довільну цілочисельну матрицю $C(m, n)$. Діапазон значень елементів матриці $[-50, 100]$. В 2, 5, 8 і т.д. стовпцях матриці (окремо) визначити максимальні по модулю елементи та їх координати. Матрицю, значення та координати максимальних елементів надрукувати.
7. Задано довільну цілочисельну матрицю $D(n, n)$. Діапазон значень елементів матриці $[-100, 50]$. Для елементів матриці, розташованих на її головній діагоналі знайти максимальний елемент серед кратних двом, та його координати. Надрукувати матрицю, максимальний елемент та його координати.
8. Задано довільну цілочисельну матрицю $G(m, n)$. Діапазон значень елементів матриці $[-100, 200]$. У кожному з парних стовпців матриці визначити суми, кількості та середні значення додатних елементів, кратних двом. Матрицю, отримані суми, кількості та середні значення надрукувати.
9. Задано довільну цілочисельну матрицю $K(m, n)$. Діапазон значень елементів матриці $[-200, 100]$. В 2, 5, 8 і т.д. стовпцях матриці (окремо) знайти максимальні елементи, кратні двом та їх координати. Надрукувати матрицю, максимальні елементи та їх координати.

10. Задано довільну цілочисельну матрицю $A(n, n)$. Діапазон значень елементів матриці $[-50, 150]$. Для елементів матриці, які знаходяться на її побічній діагоналі визначити суму, кількість та середнє значення всіх від'ємних елементів, кратних двом. Матрицю, отримані суми, кількість та середнє значення надрукувати.
11. Задано довільну цілочисельну матрицю $B(n, n)$. Діапазон значень елементів матриці $[-150, 50]$. Для елементів матриці, які знаходяться на її побічній діагоналі визначити мінімальні елементи, кратні трьом та їх координати. Матрицю, значення та координати мінімальних елементів надрукувати.
12. Задано довільну цілочисельну матрицю $Y(m, n)$. Діапазон значень елементів матриці $[-250, 200]$. В 2, 5, 8 і т.д. стовпцях матриці Y (окремо) визначити суми, кількості та середні значення від'ємних елементів, кратних трьом. Матрицю Y , отримані суми, кількості та середні значення надрукувати.
13. Задано довільну цілочисельну матрицю $Q(m, n)$. Діапазон значень елементів матриці $[-200, 400]$. У кожному непарному стовпці матриці Q визначити суми, кількості та середні значення додатних елементів кратних чотирьом. Матрицю Q , отримані суми, кількості та середні значення надрукувати.
14. Задано довільну цілочисельну матрицю $A(m, n)$. Діапазон значень елементів матриці $[-100, 100]$. У кожному непарному стовпці матриці A (окремо) визначити суми, кількості та середні значення від'ємних елементів кратних п'яти. Матрицю A , отримані суми, кількості та середні значення надрукувати.
15. Задано довільну цілочисельну матрицю $L(m, n)$. Діапазон значень елементів матриці $[-400, 400]$. У кожному парному стовпці матриці L (окремо) знайти значення та координати максимальних елементів, кратних трьом. Матрицю, значення та координати максимальних елементів надрукувати.
16. Задано довільну цілочисельну матрицю $P(m, n)$. Діапазон значень елементів матриці $[-100, 300]$. У кожному стовпці матриці окремо визначити суми, кількості та середні значення додатних елементів кратних, п'яти. Матрицю P , отримані суми, кількості та середні значення надрукувати.
17. Задано довільну цілочисельну матрицю $S(m, n)$. Діапазон значень елементів $[-150, 150]$. У 1, 4, 7 і т.д. рядках матриці S (окремо) визначити мінімальні елементи, кратні чотирьом та їх координати. Матрицю S , значення та координати мінімальних елементів надрукувати.
18. Задано довільну цілочисельну матрицю $R(m, n)$. Діапазон значень елементів матриці $[-300, 100]$. У 1, 4, 7 і т.д. рядках матриці (окремо) визначити суми, кількості та середні значення додатних елементів кратних п'яти. Матрицю R , отримані суми, кількості та середні значення надрукувати.
19. Задано довільну цілочисельну матрицю $A(m, n)$. Діапазон значень елементів матриці $[-300, 150]$. У кожному непарному рядку матриці (окремо) знайти максимальні елементи, кратні чотирьом. Надрукувати матрицю A ,

максимальні елементи та їх координати.

20. Задано довільну цілочисельну матрицю $D(n, n)$. Діапазон значень елементів матриці $[-200, 300]$. Для елементів матриці D , які знаходяться на її побічній діагоналі визначити мінімальний елемент, кратний п'яти, та його координати. Надрукувати матрицю D , значення та координати мінімального елементу.
21. Задано довільну цілочисельну матрицю $B(m, n)$. Діапазон значень елементів матриці $[-150, 250]$. У кожному стовпці матриці окремо визначити суми, кількості та середні значення від'ємних елементів кратних трьом. Матрицю B , отримані суми, кількості та середні значення надрукувати.
22. Задано довільну цілочисельну матрицю $D(m, n)$. Діапазон значень елементів матриці $[-200, 150]$. У кожному непарному стовпці матриці (окремо) визначити максимальні елементи, кратні п'яти, та їх координати. Надрукувати матрицю D , значення та координати максимальних елементів.
23. Задано довільну цілочисельну матрицю $C(m, n)$. Діапазон значень елементів матриці $[-200, 180]$. У кожному рядку матриці окремо визначити мінімальні по модулю елементи та їх координати. Матрицю C , значення та координати мінімальних елементів надрукувати.
24. Задано довільну цілочисельну матрицю $L(m, n)$. Діапазон значень елементів матриці $[-300, 200]$. У кожному парному рядку матриці (окремо) визначити суми, кількості та середні значення додатних елементів кратних, чотирьом. Матрицю L , отримані суми, кількості та середні значення надрукувати.
25. Задано довільну цілочисельну матрицю $H(m, n)$. Діапазон значень елементів матриці $[-400, 200]$. У 2, 5, 8 і т.д. стовпцях матриці H (окремо) визначити суми, кількості та середні значення додатних елементів кратних двом. Матрицю H , отримані суми, кількості та середні значення надрукувати.
26. Задано довільну цілочисельну матрицю $B(m, n)$. Діапазон значень елементів матриці $[-150, 300]$. У кожному непарному стовпці матриці (окремо) визначити суми, кількості та середні значення всіх від'ємних елементів кратних двом. Матрицю B , отримані суми, кількості та середні значення надрукувати.
27. Задано довільну цілочисельну матрицю $P(m, n)$. Діапазон значень елементів матриці $[-200, 200]$. У кожному парному стовпці матриці окремо визначити суми, кількості та середні значення додатних елементів кратних п'яти. Матрицю P , отримані суми, кількості та середні значення надрукувати.
28. Задано довільну матрицю дійсних чисел $D(m, n)$. Діапазон значень елементів матриці $[-150, 150]$. У кожному непарному рядку матриці окремо визначити мінімальні за модулем елементи та їх координати. Матрицю D , значення та координати мінімальних елементів надрукувати.
29. Задано довільну матрицю дійсних чисел $F(m, n)$. Діапазон значень елементів матриці $[-300, 300]$. У 2, 5, 8 і т.д. рядках матриці (окремо) знайти суми,

кількості та середні значення від'ємних елементів. Матрицю **F**, отримані суми, кількості та середні значення надрукувати.

30. Задано довільну матрицю дійсних чисел **A(n, n)**. Діапазон значень елементів матриці $[-100, 150]$. Для елементів матриці, які знаходяться на її побічній діагоналі визначити суму, кількість та середнє значення всіх від'ємних елементів, ціла частина яких кратна двом. Матрицю, отримані суму, кількість та середнє значення надрукувати.



Контрольні питання

- 1) Що таке вектор? Що потрібно для використання векторів у програмі?
- 2) Що таке шаблон класу? Що таке створення екземпляру шаблону? Наведіть приклади.
- 3) Наведіть способи ініціалізації об'єктів класу `vector`.
- 4) Що таке ініціалізація переліком та ініціалізація значення для об'єкту типу `vector`? У чому відмінність використання круглих та фігурних дужок під час ініціалізації об'єкту типу `vector`, наведіть приклади?
- 5) Перерахуйте основні операції з векторами.
- 6) Поясніть як можна отримати доступ до елементів вектора за допомогою індексування та ітераторів? Наведіть приклади.
- 7) Що таке псевдонім типу та які методи створення псевдонімів ви знаєте? Наведіть приклади оголошення псевдонімів.
- 8) Що таке оператор звернення до значення та оператор звернення до члену? Наведіть приклади.
- 9) Для чого потрібен оператор стрілки? Наведіть приклади.
- 10) Перерахуйте основні відмінності між масивами та векторами. Який із цих типів більш зручний у використанні?

Комп'ютерний практикум №7

Оператори

Мета: вивчити умовні оператори: оператор *if*, оператор *switch*; ітераційні оператори: цикл *do while*, серійний оператор циклу *for*; оператори переходу: *break*, *continue*. Засвоїти принципи застосування обробки виключень: робота з оператором *throw* та блоком *try*. Відпрацювати роботу зі структурами даних: використання оператора *struct*; створення та підключення файлів заголовку.

Завдання: створити консольний додаток для реалізації обчислень поставленої задачі згідно отриманого варіанту завдання.

Загальні вимоги.

- 1) Створити консольний додаток з ім'ям виду *PrizvischeKP7*.
- 2) Програмний код модуля має бути чітко структурований.
- 3) Імена об'єктів мають нести сенсові навантаження.
- 4) Програмний код має супроводжуватись коментарями в тексті програми.

Вимоги до виконання.

- 1) Передбачити обробку виключень у програмі за допомогою блоків *try* та *catch*.
- 2) Включити в програму кілька блоків *catch* для обробки різних типів виключень.
- 3) Використати оператор *throw* для передачі виключень у блоки *catch*.
- 4) Передбачити введення вхідних та виведення вихідних даних за допомогою зовнішніх файлів.
- 5) Перед введенням вхідних даних запропонувати користувачу вибір джерела введення – з зовнішнього файлу чи з клавіатури.
- 6) За допомогою оператора *switch* організувати перебір потрібних варіантів згідно свого завдання.
- 7) Застосувати оператори переходу *break* та (або) *continue* для переривання роботи оператора *switch* та операторів циклу.
- 8) Для збереження результатів розрахунку використати тип *vector* та власну структуру даних, створену у файлі заголовку.
- 9) Результати розрахунку вивести на екран та (або), на запит користувача, у зовнішній файл у режимі додавання в кінець.
- 10) За можливості компілятора використати в програмі серійний оператор циклу *for*.

- 11) По завершенню розрахунків запропонувати користувачу з застосуванням оператору циклу *do while* вибір: завершити роботу програми чи запустити її на повторний розрахунок.

Теоретичні відомості

Тема 7.1. Оператори зміни порядку виконання програми

7.1.1 Умовний оператор (?:)

Одним з різновидів умовного оператора є *умовний оператор* (?:). Він аналогічний оператору *if else*. Синтаксис оператора наступний:

умова ? вираз1 : вираз2;

Умовний оператор ? : є тернарним оператором (тобто має три операнди). Умовний оператор працює наступним чином:

- перший операнд (умова) обраховується і неявно перетворюється в *bool*;
- якщо результатом першого операнду буде значення *true* (1), обчислюється другий операнд (вираз1);
- якщо результатом першого операнду буде значення *false* (0), обчислюється третій операнд (вираз2).

Умовний оператор ? : може застосовуватись там, де використання оператора *if else* неможливе, наприклад в операторі *return* якоїсь функції

```
return ival == 10 ? 1 : 0; //if else використовувати тут не можна
```

або в операторі виводу

```
cout << (ival < 100 ? "ival<100" : "ival>=100"); //з if таке не можливе
```

чи операторі присвоєння в якості правостороннього операнда

```
double число = 16.89;  
string значення = "Вказане число ";  
значення += число < 100 ? "< 100" : ">= 100";  
cout << значення << endl;
```

Результатом роботи останнього прикладу буде:

Вказане число < 100

7.1.2 Оператор switch

Оператор `switch` надає зручний спосіб вибору однієї з кількох альтернатив. *Оператор switch* – оператор умовного виконання, котрий спочатку обраховує результат виразу, що йде за ключовим словом `switch`, а потім передає управління розділу `case`, мітка якого співпадає з результатом виразу. Коли відповідної мітки немає, виконання переходить до розділу `default` (якщо він є) або до наступного оператора, який йде за оператором `switch`, при відсутності розділу `default`. Оператор `switch` має наступний синтаксис

```
switch (вираз)
{
case мітка1: // Константний вираз 1
    вираз1; // Блок операторів 1
    break;
case мітка2: // Константний вираз 2
    вираз2; // Блок операторів 2
    break;
default:
    вираз3; // Блок операторів 3
    break;
}
```

Результат виразу, який міститься в круглих дужках після ключового слова `switch` повинен мати *цілочисельний тип* (чи можливість неявного перетворення в цілочисельний тип) або *тип enum* (тип перерахування - перелік значень, які мають чітко визначену послідовність).

В кінці кожного блоку `case` та блоку `default` необхідно застосовувати оператор `break` для переривання роботи оператора `switch`. В іншому випадку після співпадіння мітки і виконання відповідного блоку `case`, продовжиться виконання наступних блоків `case`. *Оператор break* перериває потік виконання і передає управління першому оператору після оператора `switch`.

Ключове слово `case` і пов'язане з ним значення називають також *міткою case* (`case label`). Значення кожної мітки `case` має бути *константним виразом*.

```
char ch;
int ival = 28;
switch (ch)
{
case 3.14: //Помилка! Мітка не ціле число
    //Вираз
    break;
case ival: //Помилка! Мітка не константа
```

```
        //Вираз  
        break;  
default:  
        //Вираз  
        break;  
}
```

Однакові значення міток неприпустимі.

Надалі наведений приклад програми з використанням оператора switch.

```
#include "stdafx.h"  
#include <iostream>  
using namespace std;  
#include <locale>;  
#include "windows.h";  
  
int main()  
{  
    SetConsoleCP(1251);  
    SetConsoleOutputCP(1251);  
  
    int вибір; // Змінна для вибору в операторі switch  
    double число1, число2; // Змінні для збереження операндів  
    cout << "Введіть перше число: ";  
    cin >> число1;  
    cout << "Введіть друге число: ";  
    cin >> число2;  
    cout << "Оберіть дію:\n 1 - Додавання\n 2 - Віднімання\n 3 - Множення\n 4 - Ділення\n";  
    cin >> вибір;  
    switch (вибір) // Початок оператора switch  
    {  
        case 1: // Якщо вибір = 1  
        {  
            // Виконати додавання  
            cout << число1 << " + " << число2 << " = "  
                << число1 + число2 << endl;  
            break;  
        }  
        case 2: // Якщо вибір = 2  
        {  
            // Виконати віднімання  
            cout << число1 << " - " << число2 << " = "  
                << число1 - число2 << endl;  
            break;  
        }  
        case 3: // Якщо вибір = 3  
        {  
            // Виконати множення  
            cout << число1 << " * " << число2 << " = "
```



```

        << число1 * число2 << endl;
        break;
    }
    case 4: // Якщо вибір = 4
    {
        // Виконати ділення
        cout << число1 << " / " << число2 << " = "
             << число1 / число2 << endl;
        break;
    }
    default: // Якщо вибір дорівнює будь-якому іншому значенню
        cout << "Невірний вибір!" << endl;
    }
    // Очікування натискання будь-якої клавіші
    //перед закриттям вікна консолі
    system("pause");
    return 0;
}

```

Дана програма запитує у користувача два числа, а потім пропонує обрати дію, яку необхідно виконати над цими числами. Приклад роботи даної програми наведений нижче.

```

Введіть перше число: 68
Введіть друге число: 79
Оберіть дію:
1 - Додавання
2 - Віднімання
3 - Множення
4 - Ділення
4
68 / 79 = 0.860759

```

Оператор `switch` може містити, а може і не містити зарезервоване слово `default`. Якщо значення змінної не співпадає з жодним константним виразом блоків `case` і не буде блоку `default`, то програмне керування в цьому випадку просто перейде до першого оператору після `switch`.

7.1.3 Серійний оператор `for`

Серійний оператор `for`, на відміну від традиційного оператора `for`, створений для спрощення перебирання елементів контейнеру чи іншої послідовності. Синтаксис серійного оператора `for` (range `for`) наступний:

```

for (об'явлення: вираз)
    оператор

```


Об'явлення визначає змінну. Кожен елемент послідовності повинен допускати перетворення в тип змінної. Простіше за все гарантувати відповідність типів за рахунок використання *специфікатора типу* `auto`. Так компілятор виведе тип сам. Якщо необхідний запис в елементи послідовності, то змінна циклу повинна мати *тип посилання*.

Вираз має надати деяку послідовність, таку, як список ініціалізації, масив, або об'єкт такого типу, як `vector` чи `string`, у якого є функції-члени `begin()` та `end()`, котрі повертають ітератори.

На кожній ітерації керуюча змінна визначається та ініціалізується наступним значенням послідовності, а потім виконується *оператор*. Як зазвичай, оператор може бути *одиначним оператором* або *блоком*. Виконання завершується, коли всі елементи оброблені.

В наступному прикладі цикл подвоює значення кожного елементу вектору:

```
vector<int> v = { 0,1,2,3,4,5,6,7,8,9 };  
//для запису в елементи змінна діапазону має бути посиланням  
for (auto &r : v) //для кожного елементу вектора v  
    r *= 2; //подвоїти значення кожного елементу
```

Заголовок `for` об'являє, що керуюча змінна циклу `r` пов'язана з вектором `v`. Щоб дозволити компілятору самостійно вивести тип змінної `r`, використовуємо специфікатор типу `auto`. Оскільки передбачається зміна значень елементів вектору `v`, оголошуємо змінну `r` як посилання за допомогою оператора `&`. При присвоєнні їй значень в циклі фактично присвоюється значення елементу, з яким пов'язана змінна `r` на даний момент.

З використанням традиційного циклу `for` та без застосування специфікатора `auto` попередній цикл мав би наступний вигляд:

```
for (vector<int>::iterator r = v.begin(); r != v.end(); r++)  
    *r *= 2;
```

В даному варіанті в заголовку циклу `for` визначається ітератор `r` типу `vector<int>::iterator`; задається початкове значення ітератора `r` за допомогою функції-члена `begin()`; перевіряється умова, чи не досяг ітератор кінцевого елементу вектора, за допомогою функції-члена `end()`; та задається прирощення ітератора `r` за допомогою оператору постфіксного інкременту `++`. В тілі циклу подвоюється значення поточного елементу вектора з використанням оператора звернення до значення `*` та складеного оператора присвоєння `*=`.

Якщо в заголовку серійного оператору циклу `for` замість посилання `&r` використати змінну типу значення, то елементи контейнеру змінюватись не будуть. Так, якщо в попередньому прикладі зробити цю зміну, то отримаємо програмний код, в якому ми спочатку ніби подвоюємо значення вектора та

виводимо їх на екран. Для перевірки після роботи першого циклу ще раз виводимо на екран елементи вектора.

```
#include "stdafx.h"
#include <vector>
#include <iostream>
using namespace std;

int main()
{
    vector<int> v = { 0,1,2,3,4,5,6,7,8,9 };
    for (auto r : v) { // Змінна r не є посиланням
        // Для кожного елементу вектора v
        r *= 2; // подвоїти значення кожного елементу
        cout << r << "\t"; // вивести значення на екран
    }
    cout << endl;
    for (auto &r : v) // Для кожного елементу вектора v
        cout << r << "\t"; // вивести значення на екран
    cout << endl;
    return 0;
}
```

Результатом роботи даної програми буде:

0	2	4	6	8	10	12	14	16	18
0	1	2	3	4	5	6	7	8	9

За результатами роботи першого серійного оператора `for` з використанням змінної-значення бачимо, що всі значення вектора ніби подвоїлися, як ми і очікували. Але при повторному виведенні елементів вектору на екран за допомогою іншого циклу `for`, ми помічаємо, що значення вектору не змінилися. Справа в тому, що коли ми не використовуємо посилання, змінюється лише поточне значення змінної, в якій створена копія значення елементу вектора. Сам елемент вектора при цьому не міняється.

При використанні посилання ми працюємо безпосередньо з самим елементом вектора, бо посилання це лише інше ім'я об'єкту. Якщо замінити в заголовку серійного оператора `for` змінну-значення на змінну-посилання, ми отримаємо очікуваний результат.

```
vector<int> v = { 0,1,2,3,4,5,6,7,8,9 };
//для запису в елементи змінна діапазону має бути посиланням
for (auto &r : v) {
    // Для кожного елементу вектора v
    r *= 2; // подвоїти значення кожного елементу
    cout << r << "\t"; // вивести значення на екран
}
```



```

}
cout << endl;
for (auto &r : v) // Для кожного елементу вектора v
    cout << r << "\t"; // вивести значення на екран
cout << endl;

```

Внаслідок роботи даного блоку коду на екрані побачимо наступні значення:

0	2	4	6	8	10	12	14	16	18
0	2	4	6	8	10	12	14	16	18

Тепер очевидно, що значення елементів вектора дійсно були змінені.

Серійний оператор `for` неможливо використовувати для додавання елементів у вектор чи інший контейнер. В серійному операторі `for` кешується (запам'ятовується) значення `end()`. Якщо додати чи видалити елементи з послідовності, збережене значення `end()` стане невірним.

7.1.4 Оператори переходу

Оператори переходу переривають потік виконання. Мова C++ надає чотири оператори: `break`, `continue`, `goto` та `return`.

7.1.4.1 Оператор *break*

Оператор *break* завершує найближчий оточуючий оператор `while`, `do while`, `for` або `switch`. Виконання поновлюється з оператора, що знаходиться безпосередньо за оператором, робота якого завершується.

Оператор `break` може розташовуватись лише в циклі або операторі `switch` (включно з операторами або блоками, вкладеними в ці цикли). Оператор `break` впливає лише на найближчий оточуючий цикл або оператор `switch`.

В операторі `switch` оператор `break` змушує програму виконувати наступний оператор, що знаходиться зовні оператора `switch`. Без оператора `break` виконується кожен оператор з зіставленої мітки `case` до кінця оператора `switch`, включаючи пропозицію `default`.

У циклах оператор `break` завершує виконання найближчого зовнішнього оператора `do`, `for` або `while`. Управління передається оператору, який розташований після завершення оператора, при його наявності.

Якщо використовуються вкладені оператори, `break` завершує виконання тільки того оператора `do`, `for`, `switch` або `while`, який безпосередньо його оточує. Передати управління з більш глибоко вкладених структур можна за допомогою оператора `return` або `goto`.

В наступному коді наведений приклад використання оператора `break` у циклі `for`.

// Приклад використання у стандартному циклі `for`

```
for (int i = 1; i < 10; i++) {  
    cout << i << '\t';  
    if (i == 4) {  
        cout << endl;  
        break;  
    }  
}
```

// Приклад використання у серійному циклі `for`

```
int nums[] { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };  
for (int i : nums) {  
    if (i == 4) {  
        cout << endl;  
        break;  
    }  
    cout << i << '\t';  
}
```

Результат роботи даного прикладу:

```
1      2      3      4  
1      2      3
```

В наступному коді наведено приклад використання оператора `break` у циклі `while` та циклі `do`.

```
int i = 0;  
while (i < 10) {  
    if (i == 5) {  
        cout << endl;  
        break;  
    }  
    cout << i << '\t';  
    i++;  
}
```

```
i = 0;  
do {  
    if (i == 5) {  
        cout << endl;  
        break;  
    }  
    cout << i << '\t';  
    i++;  
}
```

```
} while (i < 10);
```

Результат роботи даного прикладу:

```
0      1      2      3      4
0      1      2      3      4
```

У наступному коді показаний спосіб використання оператора `break` в операторі `switch`. Необхідно використовувати `break` завжди, коли необхідно обробити кожен випадок окремо; якщо `break` не використовується, виконання коду продовжиться до наступного випадку без завершення блоку `switch`.

```
#include "stdafx.h"
#include <iostream>
using namespace std;
#include <locale>;
#include "windows.h";

enum карти { Бубна, Чірва, Хрест, Піка };

int main()
{
    //Встановлення таблиці кодування для укр. мови
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);

    карти рука;
    рука = Чірва;
    // В цьому прикладі кожен case обробляється окремо та незалежно
    switch (рука)
    {
        case Бубна:
            cout << "У Вас в руці Бубна \n";
            break;
        case Чірва:
            cout << "У Вас в руці Чірва \n";
            break;
        case Хрест:
            cout << "У Вас в руці Хрест \n";
            break;
        case Піка:
            cout << "У Вас в руці Піка \n";
            break;
        default:
            cout << "У Вас немає карт \n";
    }
    // В цьому прикладі Бубна і Чірва обробляється одним способом,
    // а Хрест та Піка обробляється іншим способом
}
```



```
switch (рука)
{
    case Бубна:
    case Чирва:
        cout << "У Вас в руці червона карта \n";
        break;
    case Хрест:
    case Піка:
    default:
        cout << "У Вас в руці чорна карта \n";
}
return 0;
}
```

Результат роботи даного програмного прикладу буде наступний:

```
У Вас в руці Чирва
У Вас в руці червона карта
```

7.1.4.2 Оператор *continue*

Оператор *continue* перериває поточну ітерацію найближчого циклу і негайно починає наступну. Оператор *continue* може знаходитись лише в циклах *while*, *do while* або *for*, включно з операторами або блоками, вкладеними в ці цикли. Однак, на відміну від оператора *break*, оператор *continue* може бути присутнім в операторі *switch* лише як складова вбудованого ітераційного циклу.

Оператор *continue* перериває лише поточну ітерацію; виконання залишається в циклі. У випадку циклів *while* або *do while* виконання продовжується з оцінки умови. В традиційному циклі *for* виконання продовжується у виразі заголовку. В серійному операторі *for* виконання продовжується з ініціалізації керуючої змінної наступним елементом послідовності.

Наступний цикл читає зі стандартного пристрою введення по одному слову за раз. Оброблені будуть тільки ті слова, які починаються з символу підкреслення. Для любого іншого значення поточна ітерація завершується.

```
string buf;
while (cin >> buf && !buf.empty())
{
    if (buf[0] != '_')
    {
        continue; //отримати наступне значення
    }
}
```


Наступний приклад ілюструє принцип роботи оператора `continue` в циклі `do`.

```
int i = 0;
do
{
    i++;
    printf_s("Оператори циклу виконуються до оператора continue\n");
    continue;
    printf("Після оператора continue інші оператори не виконуються\n");
} while (i < 3);
printf_s("Наступні оператори після циклу\n");
```

Результат роботи даного блоку коду:

Оператори циклу виконуються до оператора `continue`
Оператори циклу виконуються до оператора `continue`
Оператори циклу виконуються до оператора `continue`
Наступні оператори після циклу

7.1.4.3 Оператор `goto`

Оператор `goto` забезпечує безумовний перехід до іншого оператора в тій же функції. Оператор `goto` ускладнює розуміння та можливість зміни програми, тому його використання не рекомендоване.

Синтаксис оператора:

```
goto мітка; // Перехід на рядок з відповідною міткою

мітка: // Рядок коду, якому передає виконання оператор goto
// оператори
```

Оператор, мітка якого задана в параметрі `мітка`, повинен знаходитися в поточній функції. Всі імена, задані в параметрі `мітка`, є членами внутрішнього простору імен `i`, отже, не перетинаються з іншими ідентифікаторами.

Мітка оператора має значення тільки для оператора `goto`; в інших випадках мітки операторів ігноруються. Повторне оголошення міток неможливе.

Замість оператора `break` рекомендується використовувати оператори `continue`, `return` і `goto`. Оскільки оператор `break` виконує вихід тільки з одного рівня циклу, можливі ситуації, коли для виходу з глибоко вкладеного циклу може знадобитися використання оператора `goto`.

У цьому прикладі оператор `goto` передає управління в точку з міткою `зупинка:`, коли значення змінної `i` дорівнює 3.


```
#include "stdafx.h"
#include <locale>;
#include "windows.h";

int main()
{
    //Встановлення таблиці кодування для укр. мови
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);

    int i, j;
    for (i = 0; i < 10; i++)
    {
        printf_s("Виконання зовнішнього циклу. i = %d\n", i);
        for (j = 0; j < 2; j++)
        {
            printf_s("\tВиконання внутрішнього циклу. j = %d\n", j);
            if (i == 3)
                goto зупинка;
        }
    }
    // Це повідомлення не виводиться, бо спрацює
    //перехід до мітки зупинка:
    printf_s("Цикли завершені. i = %d\n", i);

    зупинка:
    printf_s("Перехід на мітку зупинка:. i = %d, j = %d\n", i, j);
    return 0;
}
```

Результат роботи даного прикладу:

```
Виконання зовнішнього циклу. i = 0
    Виконання внутрішнього циклу. j = 0
    Виконання внутрішнього циклу. j = 1
Виконання зовнішнього циклу. i = 1
    Виконання внутрішнього циклу. j = 0
    Виконання внутрішнього циклу. j = 1
Виконання зовнішнього циклу. i = 2
    Виконання внутрішнього циклу. j = 0
    Виконання внутрішнього циклу. j = 1
Виконання зовнішнього циклу. i = 3
    Виконання внутрішнього циклу. j = 0
Перехід на мітку зупинка:. i = 3, j = 0
```

7.1.4.4 Оператор *return*

Оператор *return* завершує виконання функції і повертає елемент управління в зовнішню функцію, яка викликала дану функцію (або в операційну систему при передачі управління з функції *main*). Виконання поновлюється в зовнішній функції в точці відразу після виклику даної функції.

Синтаксис оператора:

return вираз;

Значення вираз, при його наявності, перетворюється в тип, вказаний в оголошенні функції, як якщо б виконувалася ініціалізація. В результаті перетворення з типу значення вираз в тип *return* функції можуть бути створені тимчасові об'єкти.

Значення вираз повертається у функцію, яка викликала дану функцію. Якщо значення вираз пропущено, то повернене значення функції не визначене. Конструктори, деструктори і функції типу *void* не можуть визначити вираз в операторі *return*. Функції всіх інших типів повинні вказати значення в операторі *return*, тип якого сумісний з типом поверненого значення даної функції.

Якщо потік елемента управління виходить з блоку, що включає визначення функції, результат буде таким же, як якщо б був виконаний оператор *return* без виразу. Це неприпустимо для функцій, оголошених як такі, що повертають значення.

Функція може мати необмежену кількість операторів *return*.

У наступному прикладі використовується вираз з оператором *return* для отримання найбільшого з двох цілих чисел.

```
#include "stdafx.h"
#include <locale>
#include "windows.h"

int найбільше(int a, int b)
{
    return (a > b ? a : b);
}

int main()
{
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);

    int число1 = 5;
    int число2 = 7;
```



```
printf_s("\n%d найбільше з двох чисел: %d та %d\n",  
        найбільше(число1, число2), число1, число2);  
}
```

Результат виконання даного прикладу:

7 найбільше з двох чисел: 5 та 7

Наступний приклад ілюструє переривання виконання функції за допомогою оператора `return`. Будь-які оператори, що розташовані після оператора `return` у функції виконуватись не будуть, але оператор `return` може розміщуватись, наприклад, в умовному операторі. Тоді після виконання певної умови буде завершено виконання функції.

```
#include "stdafx.h"  
#include <locale>  
#include "windows.h"  
  
double корінь(int a, int b)  
{  
    return sqrt(a*b);  
}  
  
int main()  
{  
    SetConsoleCP(1251);  
    SetConsoleOutputCP(1251);  
    for (int i = 0, j = 5; i < 10; i++, j--)  
    {  
        if (j >= 0)  
        {  
            printf_s("Квадратний корінь з добутку двох чисел %d  
                    та %d = %.2f\n", i, j, корінь(i, j));  
        }  
        else  
            return 0;  
    }  
}
```

Дана програма з використанням додаткової функції перемножає два числа та потім знаходить квадратний корінь з отриманого значення. Одне з чисел може прийняти від'ємне значення і при знаходженні кореня з від'ємного числа виникне помилка. Тому в операторі `if` перевіряється умова `j >= 0`. При її виконанні викликається функція `корінь(int, int)`, в іншому разі завершується виконання функції `main` за допомогою оператора `return`:

else


```
return 0;
```

Результат виконання даної програми виглядає наступним чином:

```
Квадратний корінь з добутку двох чисел 0 та 5 = 0.00
Квадратний корінь з добутку двох чисел 1 та 4 = 2.00
Квадратний корінь з добутку двох чисел 2 та 3 = 2.45
Квадратний корінь з добутку двох чисел 3 та 2 = 2.45
Квадратний корінь з добутку двох чисел 4 та 1 = 2.00
Квадратний корінь з добутку двох чисел 5 та 0 = 0.00
```

Тема 7.2. Виключення

Виключення (exception) – це анамалії часу виконання, такі як втрата підключення до бази даних або введення непередбачуваних даних, які порушують нормальне функціонування програми. Також від виключенням розуміють об'єкт системного класу чи класу користувача, який створений операційною системою чи кодом програми у відповідь на обставини, які не дозволяють подальше нормальне виконання програми. Обробка виключень в додатку дозволяє коректно вийти із скрутної ситуації.

Обробка виключень зазвичай використовується у випадку, коли деяка частина програми виявила проблему, з якою вона не може впоратись. Причому характер проблеми такий, що не дає змоги продовжити виконання частині програми, яка виявила цю проблему. В такому випадку ділянка програми, яка виявила проблему, потребує спосіб повідомити про те що сталося та про те, що ця ділянка програми не здатна продовжити виконання. Повідомивши про те що сталося, ділянка програми, яка виявила виключення, припиняє роботу.

Кожній частині програми, здатній передати виключення, відповідає інша частина, код якої здатний обробити виключення, незалежно від того, що сталося. Наприклад, якщо проблема у неприпустимому вводі, то частина обробки могла б попросити користувача ввести правильні дані.

7.2.1 Оператор throw та блок try

Виключення забезпечують взаємодію частин програми, які виявили проблему та які вирішують її. Обробка виключень у мові C++ має на увазі наступне.

- *Оператор throw* використовується частиною коду, що виявила проблему, з якою вона не може впоратись. Про оператор throw говорять, що він *передає (raise)* виключення.
- *Блок try* використовується частиною обробки виключення. Блок try починається з *ключового слова try* і завершується однією чи декількома *директивами catch (catch clause)*. Виключення, які передані з коду, що

розміщений у блоці `try`, як правило, обробляються в одному з розділів `catch`. Оскільки розділи `catch` обробляють виключення, їх називають також *обробниками виключень* (exception handler).

- Набір визначених у бібліотеці *класів виключень* (exception class) використовується для передачі інформації про те що відбулося поміж операторами `throw` і відповідними розділами `catch`.

Частина програми, що виявила виключення, використовує оператор `throw` для передачі виключення. Він складається з ключового слова `throw`, що супроводжується виразом. Вираз визначає тип виключення, яке передається. Оператор `throw`, як правило, завершується крапкою з комою, що робить його виразом.

```
if (!infile)    //Якщо не вдалося відкрити вхідний файл -
               //ініціалізувати виключну ситуацію
{
    throw std::runtime_error("Помилка відкриття вхідного файлу.\n");
    return -1;
};
```

Тип `runtime_error` є одним з типів виключень, які визначені у заголовку `stdexcept` стандартної бібліотеки. Об'єкт класу `runtime_error` слід ініціалізувати об'єктом класу `string`. Цей рядок надає додаткову інформацію про проблему.

Блок `try` має наступний синтаксис:

```
try
{
    //оператори_програми
}
catch (об'явлення_виключення)
{
    //оператори_обробника
}
catch (об'явлення_виключення)
{
    //оператори_обробника
} //...
```

Блок `try` починаєтьс з ключового слова `try`, за яким іде блок коду у фігурних дужках, у якому потенційно можуть виникнути виключення.

Блок `try` супроводжується одним чи кількома блоками `catch`. Блок `catch` складається з трьох частин: ключового слова `catch`, об'явлення (можливо, безіменного) об'єкту у круглих дужках, що зветься *об'явленням виключення* (exception declaration), і операторного блоку. Коли об'явлення виключення у

блоці catch співпадає з виключенням, виконується пов'язаний з ним блок. По завершенню виконання коду обробника управління переходить до наступного за ним оператора.

Блок *try* - це блок, який починається ключовим словом *try* і містить один або кілька розділів *catch*. Якщо код в блоці *try* передасть виняток, а один із розділів *catch* відповідає типу цього винятку, то виключення буде оброблено кодом даного обробника. В іншому випадку виключення буде оброблено в зовнішньому блоці *try*, але якщо і цього не станеться, спрацює функція *terminate()*, яка і завершить виконання програми.

```
#include <iostream>;
using namespace std;

//Необхідно для введення/виведення української мови в консолі
#include <locale>;
#include "windows.h";

int main()
{
    //Встановлення таблиці кодування для введення укр. мови
    SetConsoleCP(1251);
    //Встановлення таблиці кодування для виведення укр. мови
    SetConsoleOutputCP(1251);

    int число1;
    int число2;
    double результат;
    int лічильник = 2;

    while (лічильник--)
    {
        cout << "Введіть значення числа 1: ";
        cin >> число1;
        cout << "Введіть значення числа 2: ";
        cin >> число2;
        cout << "число 1 + число 2 = " << число1 + число2 << endl;
        cout << "число 1 / число 2 = ";
        try // Код, який може призвести до помилки розташовано тут
        {
            if (число2 == 0)
            {
                throw 123; // Генеруємо ціле число 123
            }
            // Для отримання дробового результату від ділення
            // приводимо ділене до типу с плаваючою комою
            //float(число1)
            результат = float(число1) / число2;
        }
    }
}
```



```

        cout << результат << endl;
    }
    catch (int i)// Сюди передається число 123
    {
        cout << "Помилка №" << i << " - на 0 ділити неможна!!!"
            << endl;
    }
    cout << "число1 - число2 = " << число1 - число2 << endl;
    cout << "===== " << endl << endl;
}
cout << "Програма завершила роботу!" << endl << endl;
return 0;
}

```

Результат роботи прикладу:

```

Введіть значення числа 1: 23
Введите значение числа 2: 0
число 1 + число 2 = 23
число 1 / число 2 = Помилка №123 - на 0 ділити неможна!!!
число1 - число2 = 23
=====

```

```

Введіть значення числа 1: 15
Введите значение числа 2: 7
число 1 + число 2 = 22
число 1 / число 2 = 2.14286
число1 - число2 = 8
=====

```

Програма завершила роботу!

Оператори_програми у блоці try є звичайними програмними операторами, що реалізують логіку програми. Об'явлені в блоці try змінні недоступні поза блоком, в тому числі, недоступні і в блоках catch.

7.2.2 Стандартні класи виключень

В бібліотеці C++ визначений набір класів, об'єкти яких можна використовувати для передачі повідомлень про проблеми у функціях, що визначені в стандартній бібліотеці. Ці стандартні класи виключень можуть бути також використані у програмах, створюваних розробником. Бібліотечні класи виключень визначені у чотирьох заголовках.

- В заголовку exception визначений загальний клас виключень exception. Він повідомляє лише про те, що виключення виникло, але не надає ніякої додаткової інформації.

- В заголовку `stdexcept` визначено декілька універсальних класів виключень (табл. 7.1).
- В заголовку `new` визначений клас `bad_alloc`.
- В заголовку `type_info` визначений клас виключення `bad_cast`.

В класах `exception`, `bad_alloc` та `bad_cast` визначений лише стандартний конструктор, тому неможливо ініціалізувати об'єкти цих типів.

Поведінка виключень інших типів прямо протилежна: їх можна ініціалізувати об'єктом класу `string`, однак значенням за замовчуванням їх ініціалізувати *не можна*. При створенні об'єкту виключення любого з цих типів необхідно надати ініціалізатор. Цей ініціалізатор використовується для надання додаткової інформації про помилку.

Таблиця 7.2.1 – Стандартні класи виключень заголовку `stdexcept`

Клас	Роз'яснення
<code>exception</code>	Найбільш загальний вид проблеми
<code>runtime_error</code>	Проблема, яка може бути виявлена лише під час виконання
<code>range_error</code>	Помилка часу виконання: отриманий результат перевищує припустимий діапазон значень
<code>overflow_error</code>	Помилка часу виконання: переповнення регістру при обчисленні
<code>underflow_error</code>	Помилка часу виконання: недоповнення регістру при обчисленні
<code>logic_error</code>	Помилка в логіці програми
<code>domain_error</code>	Логічна помилка: аргумент, для якого не існує результату
<code>invalid_argument</code>	Логічна помилка: непридатний аргумент
<code>length_error</code>	Логічна помилка: спроба створити об'єкт більшого розміру, ніж максимально припустимий для даного типу
<code>out_of_range</code>	Логічна помилка: значення поза межами допустимого діапазону

У класах виключень визначена лише одна функція `what()`. Вона не отримує ніяких аргументів і повертає константний покажчик на тип `char`. Це покажчик на символьний рядок у стилі C, який містить текст опису переданого виключення.

```
try
{
    //оператори_програми
}
catch (const std::runtime_error err)           //Перехоплення виключень
{
    cout << "Виникла помилка в процесі виконання програми." << endl;
    cout << err.what() << endl;
}
catch (const std::out_of_range err)           //Перехоплення виключень
{
    cout << "Виникла помилка. Значення поза межами діапазону." << endl;
```



```
    cout << err.what() << endl;  
}
```

У кожному додатку має бути передбачена обробка виключних ситуацій для забезпечення стабільної, безвідмовної роботи.

C++

Приклад програмної реалізації

Приклад роботи з умовними операторами, операторами циклу та переходу, а також оператором throw та блоком try

```
//Пошук та заміна літер у введених словах
//

#include "stdafx.h"
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <stdexcept>
//Необхідно для виведення в консолі української мови
#include "windows.h"
//Підключення файлу заголовку зі створеною структурою даних
#include "strVec.h"
using std::cin;
using std::cout;
using std::endl;
using std::string;
using std::vector;
using std::ofstream;
using std::ifstream;

int main()
{
    //Необхідно для введення в консолі української мови
    SetConsoleCP(1251);
    //Необхідно для виведення в консолі української мови
    SetConsoleOutputCP(1251);
    //Відкриття файлу для додавання в кінець
    ofstream outfile("out_file.txt", ofstream::app);
    ifstream infile("in_file.txt");
    char n;
    do
    {
        try
        {
            string text, line, wrd, wrd123;
            strVec sVec, sVec123;
            vector<strVec> vect;
            //Запит на запуск повторного розрахунку
            cout << "Задати речення з вхідного файлу? [y] [n]"
                  << endl;
            //Якщо не вдалося відкрити вихідний файл -
            //ініціалізувати виключну ситуацію
            if (!outfile)
            {

```



```
        throw std::runtime_error("Помилка відкриття
                                вихідного файлу.\n");
    return -1;
};
cin >> n;
if (n=='y')
{
    if (!infile)    //Якщо не вдалося відкрити вхідний
                  // файл -ініціалізувати виключну ситуацію
    {
        throw std::runtime_error("Помилка відкриття
                                вхідного файлу.\n");
        return -1;
    };
    while (getline(infile, line))    //Зчитуємо текст
    //рядками в змінну text поки не буде введено ctrl+z
    {
        text += line + " ";    //Записуємо всі введені
                              //символьні рядки в змінну text
    }
}
else
{
    cout << "Задайте речення. Завершення вводу ctrl+z"
         << endl;

    while (getline(cin, line)) //Зчитуємо текст рядками
    //в змінну text поки не буде введено ctrl+z
    {
        text += line + " ";    //Записуємо всі введені
                              //символьні рядки в змінну text
    }
}
//Розбиваємо текст на слова
for (string::size_type i = 0; i != text.size(); i++)
{
    //Перевіряємо чи поточний символ графічний
    if (isgraph(text[i]))
    {
        //Додаємо поточну літеру в змінну wrd
        wrd += text[i];
        //Шукаємо потрібні літери, рахуємо їх
        //кількість та замінюємо на цифри

        switch (text[i])
        {
            case 'a':    //якщо поточна літера а
                        //лічильник літер а збільшеємо на 1
                        ++sVec.count_a;
        }
    }
}
```

```
        text[i] = '1'; //заміна літери а на 1
        break;       //переривання роботи
                    //оператора switch
    case 'o':
        ++sVec.count_b;
        text[i] = '2';
        break;
    case 'e':
        ++sVec.count_c;
        text[i] = '3';
        break;
    default:
        ++sVec.count_n;
        break;
    }
    wrd123+= text[i];    //Додаємо поточну літеру
                        //з урахуванням заміни в змінну wrd123
}
else if (wrd.size() != 0) //Якщо слово
//закінчилось, додаємо його окремим елементом
//у змінну sVec.word
{
    sVec.word=wrd; //Запам'ятовуємо слово без
                  //замін літер у змінну sVec.word
    wrd = "";
    sVec.word123 = wrd123;    //Запам'ятовуємо
                            //слово після замін літер
                            //у змінну sVec.word123
    wrd123 = "";
    vect.push_back(sVec); //Додаємо у вектор
                          //змінну типу створеної структури даних,
                          //яка містить вхідне та вихідне
                          //слово та підраховану кількість літер
    sVec.count_a = 0;
    sVec.count_b = 0;
    sVec.count_c = 0;
    sVec.count_n = 0;
}
else if (text[i] < -1 || text[i] > 255) //Діапазон
                                        //латинських літер
                                        //від -1 до 255
{
    throw std::out_of_range("Речення треба вводити
                             латинськими літерами. \n");
    return -1;
}
}
cout << "Програма підраховує кількість літер а, о, е
(та всіх інших) у слові та замінює їх відповідно
```



```
        на 1, 2, 3" << endl;
//Цикл для виведення на друк результатів розрахунку
//для кожного слова,
//які збережені у векторі
for (vector<strVec>::iterator i = vect.begin();
     i != vect.end(); i++)
{
    sVec123 = *i;
    cout << sVec123.word << ", слово після заміни "
         << sVec123.word123 << ", кількість a="
         << sVec123.count_a << ", кількість o="
         << sVec123.count_b << ", кількість e="
         << sVec123.count_c << ", кількість інших
         літер n=" << sVec123.count_n << endl;
}
cout << endl << "Вивести результати роботи програми
у текстовий файл? [y] [n]"
    << endl; //Запит про необхідність запису
            //результатів у зовнішній файл

char out;
cin.clear();
cin >> out;
for (auto &i : vect) //Серійний цикл for для виведення
                    //у зовнішній файл результатів
                    //розрахунку для кожного слова,
                    //які збережені у векторі
{
    //Переривання роботи циклу for, якщо користувач
    //обрав "n"
    if (out == 'n') break;
    sVec123 = i;
    outfile << sVec123.word << ", слово після заміни "
         << sVec123.word123 << ", кількість a="
         << sVec123.count_a << ", кількість o="
         << sVec123.count_b << ", кількість e="
         << sVec123.count_c
         << ", кількість інших літер n="
         << sVec123.count_n << endl;
}
}
//Перехоплення виключень
catch (const std::runtime_error err)
{
    cout << "Виникла помилка в процесі виконання програми."
         << endl;
    cout << err.what() << endl;
}
//Перехоплення виключень
catch (const std::out_of_range err)
```

```
{  
    cout << "Виникла помилка. Значення поза межами  
           діапазону." << endl;  
    cout << err.what() << endl;  
}  
//Запит на запуск повторного розрахунку  
cout << "Повторити введення? [y] [n]" << endl;  
cin.clear(); //Очищення потоку введення  
cin >> n;  
} while (n == 'y');  
infile.close(); // закриваємо файл  
outfile.close(); // закриваємо файл  
return 0;  
}
```

C++

Приклад оголошення нової структури даних у файлу заголовку.

//strVec.h Файл заголовку з оголошенням нової структури даних

```
#ifndef STRVEC_H
#define STRVEC_H
#include <string>
struct strVec //Створення власної структури даних (класу)
{
    std::string word;
    std::string word123;
    unsigned count_a=0;
    unsigned count_b=0;
    unsigned count_c=0;
    unsigned count_n=0;
};
#endif
```

C++

Завдання до комп'ютерного практикуму №7

1. Задати довільне речення з клавіатури чи зовнішнього файлу. Створити власну структуру даних `wordCount` з двома полями: `word` та `wordn` типу `string`. Створити власну структуру даних `count_234n` із полями `count2`, `count3`, `count4` та `countn` типу `unsigned`. Створити вектор `vecCount` типу `vector<wordCount>`. Записати всі слова окрім пробілів і знаків пунктуації у поле вектору `vecCount.word`. Підрахувати окремо кількість слів у яких 2, 3 та 4 символи, а також всі інші слова та записати отримані значення у поля `count_234n.count2`, `count_234n.count3`, `count_234n.count4` та `count_234n.countn` відповідно. В кінець кожного слова – елементу вектора, додати цілочисельне значення, яке відповідає кількості символів у слові (наприклад, «слово5»). Записати слова після додавання в кінець кожного слова кількості літер у поле вектору `vecCount.wordn`. Вивести на екран та (або) у зовнішній файл введене речення, а також кожне слово в окремий рядок до додавання кількості символів та після. Вивести кількість слів з 2-ма, 3-ма та 4-ма символами, а також всіх інших слів.
2. Задати довільне речення з клавіатури чи зовнішнього файлу. Підрахувати кількість символів в кожному слові. Створити власну структуру даних `vecStr`, яка складатиметься з чотирьох векторів типу `vector<string>`. Слова з кількістю літер 3 записати у поле вектор `vecStr.vec3`, слова з кількістю літер 4 записати у вектор `vecStr.vec4`, слова з кількістю літер 5 записати у вектор `vecStr.vec5`, всі інші слова записати у вектор `vecStr.vecn`. Вивести сформовані вектори та кількість елементів у них і початкове речення на екран та (або) у зовнішній файл.
3. Задати довільне речення з клавіатури чи зовнішнього файлу. Створити власну структуру даних `word_replc` з трьома полями: `word` та `word_aie` типу `string`, та полем `replc_count` типу `unsigned`. Створити вектор `vec_replc` типу `vector<word_replc>`. Записати всі слова окрім пробілів і знаків пунктуації у поле вектору `vec_replc.word`. Знайти у кожному слові літери «а», «і» та «е» та замінити їх на великі. Записати слова після замін у поле вектору `vec_replc.word_aie`. Підрахувати кількість замін у кожному слові та записати у поле `vec_replc.replc_count`. Вивести на екран та (або) у зовнішній файл введене речення, а також кожне слово в окремий рядок до заміни літер «а», «і» та «е» та після, а також кількість зроблених у кожному слові замін.
4. Задати довільне речення з клавіатури чи зовнішнього файлу. Записати всі слова окрім пробілів і знаків пунктуації у вектор. Створити власну структуру даних з чотирма полями: для запису слова (типу `string`), кількості літер «k» (типу `unsigned`), кількості літер «i» (типу `unsigned`) та кількості літер «c» (типу `unsigned`). Створити вектор типу створеної структури даних. Записати у

- створений вектор кожне слово та кількість знайдених в ньому літер «к», «і» та «с». Вивести на екран та (або) у зовнішній файл введене речення, а також кожне слово в окремий рядок та вказати знайдену у ньому кількість літер «к», «і» та «с».
5. Задати довільне речення з клавіатури чи зовнішнього файлу. Створити власну структуру даних `char_123n` із чотирма полями: `word` типу `vector<string>`, `char_1`, `char_2`, `char_3` типу `vector<char>` та `str_n` типу `vector<string>`. Створити змінну `vec_123n` типу створеної структури даних `char_123n`. Записати всі слова окрім пробілів і знаків пунктуації у вектор `vec_123n.word`. Сформувати вектор `vec_123n.char_1` із перших літер всіх слів, вектор `vec_123n.char_2` із других літер всіх слів, вектор `vec_123n.char_3` із третіх літер всіх слів та вектор `vec_123n.str_n`, елементами якого будуть залишки слів без перших трьох літер. Вивести на екран та (або) у зовнішній файл введене речення, а також всі створені вектори.
 6. Задати довільне речення з клавіатури чи зовнішнього файлу. Створити власну структуру даних `rplcmnt` із двома полями `word` та `wordrplc` типу `string`. Створити вектор `vecrplc` типу створеної структури даних `vector<rplcmnt>`. Записати всі слова окрім пробілів і знаків пунктуації у поле вектору `vecrplc.word`. Знайти у кожному слові літеру «j» та замінити її на «1», літеру «i» та замінити її на «2», літеру «k» та замінити її на «3», інші літери в слові змінити на великі. Записати кожне слово після заміни символів у поле вектору `vecrplc.wordrplc`. Вивести на екран та (або) у зовнішній файл введене речення, а також кожне слово в окремий рядок до заміни літер «j», «i» та «k» та після заміни їх на «1», «2» та «3».
 7. Задати довільне речення з клавіатури чи зовнішнього файлу. Створити власну структуру даних `struc_char` із шістьма полями: `word` та `str_n` типу `string` і полями `char_count`, `char_1`, `char_n`, `char_n1` типу `unsigned`. Створити вектор `vec_char` типу `vector<struc_char>`. Записати всі слова окрім пробілів і знаків пунктуації у поле вектору `vec_char.word`. Сформувати вектор `vec_char.char_1` із перших літер всіх слів, вектор `vec_char.char_n` з останніх літер всіх слів, вектор `vec_char.char_n1` із передостанніх літер всіх слів та вектор `vec_char.str_n`, елементами якого будуть залишки слів без першої та двох останніх літер. Записати в поле `vec_char.char_count` кількість символів у кожному введеному користувачем слові. Вивести на екран та (або) у зовнішній файл введене речення, а також в окремих рядках поля створеного вектору - слово, кількість у ньому літер, залишок слова без першої та двох останніх літер, першу літеру слова, останню літеру слова та передостанню літеру слова.
 8. Задати довільне речення з клавіатури чи зовнішнього файлу. Створити власну структуру даних `replacmnt` із двома полями `word` та `word13n` типу `string`.

Створити вектор `vecStrc` типу створеної структури даних `vector<replacmnt>`. Записати всі слова окрім пробілів і знаків пунктуації у поле вектору `vecStrc.word`. У кожному слові замінити першу, третю та останню літеру на велику, інші літери в слові змінити на «0». Записати слова після заміни у поле вектору `vecStrc.word13n`. Вивести на екран та (або) у зовнішній файл введене речення, а також кожне слово в окремий рядок до заміни літер та після заміни їх.

9. Задати довільне речення з клавіатури чи зовнішнього файлу. Створити власну структуру даних `wrd_cnt` із двома полями: `wrd` та `n_wrd` типу `string`. Створити власну структуру даних `cnt_135n` із чотирма полями: `cnt_1`, `cnt_3`, `cnt_5` та `cnt_n` типу `unsigned`. Створити вектор `vec_cnt` типу `vector<wrd_cnt>`. Записати всі слова окрім пробілів і знаків пунктуації у поле вектору `vec_cnt.wrd`. Створити змінну `var_135n` типу створеної структури даних `cnt_135n`. Підрахувати окремо кількість слів в яких 1, 3 та 5 символів, а також всі інші слова та записати отримані значення у поля `var_135n.cnt_1`, `var_135n.cnt_3`, `var_135n.cnt_5` та `var_135n.cnt_n` відповідно. На початок кожного слова – елементу вектора, додати цілочисельне значення, яке відповідає кількості символів у слові (наприклад, «5_слово») та записати його у поле вектору `vec_cnt.n_wrd`. Вивести на екран та (або) у зовнішній файл введене речення, а також кожне слово в окремий рядок до додавання кількості символів та після. Вивести кількість слів із 1-им, 3-ма та 5-ма символами, а також всіх інших слів.
10. Задати довільне речення з клавіатури чи зовнішнього файлу. Створити власну структуру даних `text_rty` із полями `word` та `bWord` типу `string` та полями `char_r`, `char_t`, `char_y`, `char_else` типу `unsigned`. Створити вектор `vect_rty` типу `vector<text_rty>`. Підрахувати окремо кількість символів «r», «t», «y» та всіх інших у кожному слові і записати їх у поля вектору `vect_rty.char_r`, `vect_rty.char_t`, `vect_rty.char_y` та `vect_rty.char_else` відповідно. Замінити перші літери кожного слова на великі та записати слова після заміни у поле вектору `vect_rty.bWord`. Вивести на екран та (або) у зовнішній файл введене речення, а також в окремих рядках кожне слово до, та після заміни літер з вказаною після нього підрахованою кількістю літер «r», «t», «y» та всіх інших у кожному слові.
11. Задати довільне речення з клавіатури чи зовнішнього файлу. Створити власну структуру даних `word_rplc` з трьома полями: `word` та `word_n` типу `string`, та полем `word_cnt` типу `unsigned`. Створити вектор `vec_rplc` типу `vector<word_rplc>`. Записати всі слова окрім пробілів і знаків пунктуації у поле вектору `vec_rplc.word`. Замінити в кожному слові 1-шу, 3-тю та 5-ту літери порядковим номером кожної заміни у всьому введеному реченні (наприклад, «Замінити певні літери у кожному слові» - «1a2i3ити», «4e5н6», «7i8e9и», «10», «11o12н13му», «14л15в16»). Записати слова після замін у поле вектору

vec_rplc.word_n. Підрахувати кількість замін у кожному слові та записати у поле вектору vec_rplc.word_cnt. Вивести на екран та (або) у зовнішній файл введене речення, а також кожне слово в окремий рядок до заміни та після, та кількість замін у кожному слові. Вивести загальну кількість замін в усьому реченні.

12. Задати довільний текст із клавіатури чи зовнішнього файлу, що складається з кількох речень зі знаками пунктуації. Створити власну структуру даних punct_struct, яка складатиметься з наступних полів: capitl типу vector<string>; soma_c, dot_c та punct_c типу string::size_type. Підрахувати кількість слів, які починаються з великої літери, та записати їх у поле вектор capitl. Підрахувати у тексті кількість ком, записати її у поле soma_c та замінити коми на крапку з комою. Підрахувати у тексті кількість крапок, записати її у поле dot_c та замінити крапки на двокрапку. Підрахувати інші знаки пунктуації та записати їх кількість у поле punct_c. Вивести на екран та (або) у зовнішній файл введений текст, текст після заміни ком і крапок, а також кількість ком, кількість крапок та кількість інших знаків пунктуації в тексті.
13. Задати довільне речення з клавіатури чи зовнішнього файлу, яке складається зі слів записаних великими літерами. Створити структуру даних struct_cnt із наступними полями даних: word та word_sml типу string; a_cnt, o_cnt, e_cnt та n_cnt типу unsigned. Створити вектор vect_oae типу vector<struct_cnt>. Записати всі слова окрім пробілів і знаків пунктуації у поле вектору vect_oae.word. Підрахувати кількість літер «А», «О», «Е» та замінити їх на малі літери, а також окремо підрахувати кількість всіх інших літер у кожному слові та записати їх у поля вектору vect_oae.a_cnt, vect_oae.o_cnt, vect_oae.e_cnt і vect_oae.n_cnt відповідно. Слова після заміни літер записати у поле вектору vect_oae.word_sml. Вивести на екран та (або) у зовнішній файл введене речення, а також в окремих рядках кожне слово до та після заміни літер із вказаною після нього підрахованою кількістю літер «А», «О», «Е» та всіх інших літер у кожному слові
14. Задати довільне речення з клавіатури чи зовнішнього файлу. Створити власну структуру даних repl_struc із наступними полями: wrd, wrd_13n та wrd_pair типу string; count_pair типу string::size_type. Створити вектор vect_repl типу vector<repl_struc>. Записати всі слова окрім пробілів і знаків пунктуації у поле вектору vect_repl.wrd. Замінити першу, третю та останню літеру в кожному слові на велику і записати слова після замін у поле вектору vect_repl.wrd_13n та підрахувати загальну кількість таких замін у реченні. Всі парні літери кожного слова замінити на 0 і записати слова після замін у поле вектору vect_repl.wrd_pair та підрахувати їх кількість у кожному слові і зберегти у полі vect_repl.count_pair. Вивести на екран та (або) у зовнішній файл введене речення, а також кожне слово в окремий рядок до заміни літер та після заміни

їх, кількість парних літер у кожному слові та загальну кількість заміन у реченні 1-их, 3-їх та останніх літер.

15. Задати довільне речення з клавіатури чи зовнішнього файлу. Створити власну структуру даних `sentence_strc`, яка міститиме поля: `sentnc`, `sentnc_a`, `sentnc_o` та `sentnc_e` типу `string`; `count_a`, `count_o` та `count_e` типу `string::size_type`; `vec_wrd` типу `vector<string>`. Зберегти копії речення у полях структури `sentnc`, `sentnc_a`, `sentnc_o` та `sentnc_e`. Записати всі слова окрім пробілів і знаків пунктуації у поле вектор `vec_wrd`. Підрахувати у реченні кількість літер «а», «о» та «е» та зберегти знайдені значення у поля `count_a`, `count_o` та `count_e`. Замінити кожну літеру «а» у полі `sentnc_a` цілочисельним значенням, що відповідає порядковому номеру літери «а» у реченні (наприклад, «Замінити літери 'а' порядковими значеннями їх у реченні» – «Замінити літери '2' порядковими значеннями їх у реченні»). Вивести на екран та (або) у зовнішній файл введене речення, а також речення після заміни літер «а». Аналогічно замінити літери «о» у полі `sentnc_o` та «е» у полі `sentnc_e`, та вивести на екран речення після заміни. Вивести слова з яких складається речення за допомогою поля-вектору `vec_wrd`.
16. Задати довільний текст з клавіатури чи зовнішнього файлу, який складається з кількох речень зі знаками пунктуації. Створити власну структуру даних `struct_aoe`, що складається з наступних полів: `wrд`, `wrд_big` та `wrд_repl` типу `string`; `count_little`, `count_little_a`, `count_little_o` та `count_little_e` типу `string::size_type`. Створити вектор `replvec` типу `vector<struct_aoe>`. Записати всі слова окрім пробілів і знаків пунктуації у поле вектору `replvec.wrд`. Замінити в кожному слові останні літери на великі та зберегти слова у поля вектору `replvec.wrд_big`. Підрахувати в кожному слові кількість малих літер та записати у поле вектору `replvec.count_little`. Замінити малі літери «а» на 1, малі літери «о» на 2, а малі літери «е» на 3 і зберегти слова після заміни у полях вектору `replvec.wrд_repl`. Підрахувати в кожному слові кількість літер «а», «о» і «е» та записати її у поля вектору `replvec.count_little_a`, `replvec.count_little_o` і `replvec.count_little_e` відповідно. Вивести на екран та (або) у зовнішній файл введене речення, а також в окремому рядку слово до заміни літер, після заміни останніх літер та після заміни літер «а», «о» і «е» з вказуванням кількості малих літер у слові, та окремо малих літер «а», «о» і «е».
17. Задати довільне речення з клавіатури чи зовнішнього файлу, яке складається зі слів записаних великими літерами. Створити власну структуру даних `struct_aoi`, що складається з наступних полів: `wrд`, `wrд_repl` типу `string`; `acount`, `ocount`, `icount` і `ncount` типу `string::size_type`. Створити вектор `replvec` типу `vector<struct_aoi>`. Записати всі слова окрім пробілів і знаків пунктуації у поле вектору `replvec.wrд`. Підрахувати кількість літер «А», «О» та «І» в кожному слові: записати підраховану кількість у поля `replvec.acount`, `replvec.ocount` та

replvec.icount відповідно; замінити знайдені літери на малі літери і зберегти отримане слово у поле вектору replvec.wrd_repl. Окремо підрахувати кількість всіх інших літер у кожному слові і записати її у поле replvec.ncount. Вивести на екран та (або) у зовнішній файл введене речення, а також в окремі рядки слово до та після заміни літер, кількість у слові літер «А», «О», «І» та всіх інших літер.

18. Задати довільне речення з клавіатури чи зовнішнього файлу. Створити власну структуру даних struct_char, що складається з наступних полів: wrd, wrd_char та wrd_big типу string; pos_l1, pos_l2 та pos_l3 типу string::size_type. Створити вектор replvec типу vector<struct_char>. Записати всі слова окрім пробілів і знаків пунктуації у поле вектору replvec.wrd. Ввести три різних літери l1, l2, l3, які необхідно знайти у словах. Знайти першу позицію, в якій зустрічається кожна з трьох літер у слові і записати позиції літер у поля replvec.pos_l1, replvec.pos_l2 і replvec.pos_l3 відповідно. Замінити першу знайдену літеру l1 на одиницю, першу знайдену літеру l2 на двійку та першу знайдену літеру l3 на трійку і зберегти слова після змін у поле вектору replvec.wrd_char. Замінити в кожному слові першу, другу та останню літери на великі і зберегти слова після змін у поле вектору replvec.wrd_big. Вивести на екран та (або) у зовнішній файл введене речення, а також у окремих рядках вхідне слово, слово після заміни літер l1, l2, l3 та слово після заміни літер на великі. Вказати позицію в слові, в якій вперше зустрілась кожна з літер l1, l2, l3.
19. Задати довільне речення з клавіатури чи зовнішнього файлу. Створити власну структуру даних swops_strc, яка міститиме поля: swops, swops_i, swops_o та swops_e типу string; count_i, count_o та count_e типу string::size_type; vec_wrd типу vector<string>. Зберегти копії речення у полях структури swops, swops_i, swops_o та swops_e. Записати всі слова окрім пробілів і знаків пунктуації у поле вектор vec_wrd. Підрахувати у реченні кількість літер «і», «о» та «е» та зберегти знайдені значення у поля count_i, count_o та count_e. Замінити кожен літеру «і» у полі swops_i цілочисельним значенням, що відповідає порядковому номеру літери «і» в реченні (наприклад, «Замінити літери 'і' порядковими значеннями їх у реченні» – «Зам4нити л11тери '18' порядковими значеннями їх у реченн55»). Вивести на екран та (або) у зовнішній файл введене речення, а також речення після заміни літер «і». Аналогічно замінити літери «о» у полі swops_o та «е» у полі swops_e, та вивести на екран речення після замін. Вивести слова з яких складається речення за допомогою поля-вектору vec_wrd.
20. Задати довільне речення з клавіатури чи зовнішнього файлу. Створити власну структуру даних repl_struc із наступними полями: wrd, wrd_13n та wrd_pair типу string; count_pair типу string::size_type. Створити вектор vect_repl типу vector<repl_struc>. Записати всі слова окрім пробілів і знаків пунктуації у поле

вектору `vect_repl.wrd`. Замінити другу, четверту та останню літеру в кожному слові на велику і записати слова після замін у поле вектору `vect_repl.wrd_13n` та підрахувати загальну кількість таких замін у реченні. Всі непарні літери кожного слова замінити на «1» і записати слова після замін у поле вектору `vect_repl.wrd_pair` та підрахувати їх кількість в кожному слові і зберегти у полі `vect_repl.count_pair`. Вивести на екран та (або) у зовнішній файл введене речення, а також кожне слово в окремий рядок до заміни літер та після заміни їх, кількість непарних літер у кожному слові та загальну кількість замін у реченні 2-их, 4-их та останніх літер.

21. Задати довільне речення з клавіатури чи зовнішнього файлу, яке складається зі слів записаних малими літерами. Створити власну структуру даних `struct_my`, що складається з наступних полів: `wrd`, `wrd_repl` типу `string`; `a_cnt`, `o_cnt`, `i_cnt` і `n_cnt` типу `string::size_type`. Створити вектор `chng` типу `vector<struct_my>`. Записати всі слова окрім пробілів і знаків пунктуації у поле вектору `chng.wrd`. Підрахувати кількість літер «а», «о» та «і» в кожному слові: записати підраховану кількість у поля `chng.a_cnt`, `chng.o_cnt` та `chng.i_cnt` відповідно; замінити знайдені літери на великі літери і зберегти отримане слово у поле вектору `chng.wrd_repl`. Окремо підрахувати кількість всіх інших літер у кожному слові і записати її у поле `chng.n_cnt`. Вивести на екран та (або) у зовнішній файл введене речення, а також в окремі рядки слово до та після заміни літер, кількість у слові літер «а», «о», «і» та всіх інших літер.
22. Задати довільне речення з клавіатури чи зовнішнього файлу. Підрахувати кількість символів у кожному слові, загальну кількість символів, та кількість літер (без пробілів та знаків пунктуації). Створити власну структуру даних `vecStr`, яка складатиметься з чотирьох векторів типу `vector<string>`. Слова з кількістю літер від 1 до 3 записати у поле вектор `vecStr.vec3`, слова з кількістю літер від 4 до 6 записати у вектор `vecStr.vec4`, слова з кількістю літер від 7 до 9 записати у вектор `vecStr.vec7`, всі інші слова записати у вектор `vecStr.vecn`. Вивести сформовані вектори та кількість елементів у них і початкове речення на екран та (або) у зовнішній файл.
23. Задати довільне речення з клавіатури чи зовнішнього файлу. Створити власну структуру даних `repl_struc` із наступними полями: `wrd`, `wrd_13n` та `wrd_pair` типу `string`; `count_pair` типу `string::size_type`. Створити вектор `vect_repl` типу `vector<repl_struc>`. Записати всі слова окрім пробілів і знаків пунктуації у поле вектору `vect_repl.wrd`. Замінити першу, третю та останню літеру в кожному слові на порядковий номер літери у слові і записати слова після замін у поле вектору `vect_repl.wrd_13n` та підрахувати загальну кількість таких замін у реченні. Всі парні літери кожного слова замінити на великі і записати слова після замін у поле вектору `vect_repl.wrd_pair` та підрахувати їх кількість у кожному слові і зберегти у полі `vect_repl.count_pair`. Вивести на екран та (або)

у зовнішній файл введене речення, а також кожне слово в окремий рядок до заміни літер та після заміни їх, кількість парних літер у кожному слові та загальну кількість заміни у реченні 1-их, 3-іх та останніх літер.

24. Задати довільний текст із клавіатури чи зовнішнього файлу, що складається з кількох речень зі знаками пунктуації. Створити власну структуру даних `punct_struct`, яка складатиметься з наступних полів: `cptl` типу `vector<string>`; `com_c`, `dot_c` та `punct_c` типу `string::size_type`. Підрахувати кількість слів, які починаються з великої літери, замінити перші літери на малі та записати їх у поле вектор `cptl`. Підрахувати у тексті кількість ком, записати її у поле `com_c` та замінити коми на крапку з комою. Підрахувати у тексті кількість крапок, записати її у поле `dot_c` та замінити крапки на двокрапку. Підрахувати інші знаки пунктуації та записати їх кількість у поле `punct_c`. Вивести на екран та (або) у зовнішній файл введений текст, текст після заміни ком і крапок, а також кількість ком, кількість крапок та кількість інших знаків пунктуації в тексті.
25. Задати довільне речення з клавіатури чи зовнішнього файлу. Створити власну структуру даних `rplcmnt` із двома полями `word` та `wordrplc` типу `string`. Створити вектор `vesrplc` типу створеної структури даних `vector<rplcmnt>`. Записати всі слова окрім пробілів і знаків пунктуації у поле вектору `vesrplc.word`. Знайти у кожному слові літеру «l», літеру «o», літеру «k» та замінити їх на великі, інші літери в слові змінити на малі. Записати кожне слово після заміни символів у поле вектору `vesrplc.wordrplc`. Вивести на екран та (або) у зовнішній файл введене речення, а також кожне слово в окремий рядок до заміни літер «l», «o» та «k» та після заміни їх.

Контрольні питання

- 1) Що таке оператор? Що таке оператор керування потоком виконання? Що таке оператор виразу?
- 2) Що таке пустий оператор? Які особливості його використання?
- 3) Що таке складений оператор? Що таке область видимості блоку, область видимості керуючої структури?
- 4) Що таке умовний оператор (`?:`), наведіть приклад його використання.
- 5) Що таке умовний оператор `switch`? Що таке блок `case` та блок `default`, для чого вони використовуються?
- 6) Що таке оператор `break`? Які особливості використання міток `case`? Наведіть приклад використання оператора `switch`?
- 7) Що таке ітераційні оператори? Що таке цикли з передумовою та післяумовою?
- 8) Що таке серійний оператор `for`? Опишіть принцип його роботи. В чому відмінність серійного оператора `for` від традиційного?
- 9) Що таке оператор `do while`? В чому його відмінність від оператора `while`? Наведіть приклад організації програми з можливістю повторної організації розрахунків по запиту користувача.
- 10) Опишіть особливості використання операторів `break` та `continue`, в чому їх відмінність?
- 11) Що таке виключення? Що таке обробка виключень?
- 12) Що таке оператор `throw`? Що таке блок `try`? Що таке розділ `catch`?
- 13) Які бібліотечні класи виключень ви знаєте, в яких заголовках вони визначені? Перерахуйте стандартні класи виключень заголовку `stdexcept`?
- 14) Для чого потрібна функція `what()`? Наведіть приклад її використання.
- 15) Як організувати власну структуру даних? Як створити файл заголовку для оголошення класу?

Комп'ютерний практикум №8

Функції

Мета: вивчити принципи створення *функцій користувача* та роботи з ними: *визначення функції*; *тіло функції*; тип значення, яке повертає функція з використанням оператора *return*; *оголошення функцій у файлі заголовку*; виклик функції; передача аргументів; створення програм за допомогою функцій.

Завдання: створити консольний додаток для обчислення значень функціональної залежності згідно отриманого варіанту завдання.

Загальні вимоги.

- 1) Створити консольний додаток з ім'ям виду *PrizvischeKP8*.
- 2) Програмний код модуля має бути чітко структурований.
- 3) Імена об'єктів мають нести сенсові навантаження.
- 4) Програмний код має супроводжуватись коментарями в тексті програми.

Вимоги до виконання.

- 1) Створити за допомогою *файлу заголовку* власну *структуру даних* *structf*, яка зберігатиме розраховане значення функції та значення аргументів, від яких вона залежить.
- 2) Створити *вектор*, типом значень якого буде створена *структура даних* *vector<structf>*.
- 3) Запрограмувати *функцію користувача* *functf* у зовнішньому файлі *functf.cpp*, яка розраховуватиме необхідне значення функції на певній ітерації.
- 4) У якості *параметрів функції* *functf* використати *параметри посилання*.
- 5) Записувати розраховане значення функції на кожній ітерації та аргументи функції, як окремий елемент створеного *вектору* типу *vector<structf>*.
- 6) Створити *функцію користувача* *prntf* у зовнішньому файлі *functf.cpp*, яка буде виводити на екран та у зовнішній текстовий файл елементи *вектору* (розраховане значення функції та її аргументи).
- 7) В якості *параметрів функції* *prntf* використати *константні посилання*.
- 8) Об'явити у функції *prntf* локальну *статичну змінну* типу *static int* для підрахунку ітерацій та передбачити виведення номеру ітерації на екран та у зовнішній файл разом зі значеннями функції.

- 9) Об'явити створені *функції користувача* `functf` та `prntf` у *файлу заголовку*. Підключити створений *файл заголовку* до файлів коду з функцією `main` та функціями `functf` і `prntf`.
- 10) Передбачити перехоплення помилки відкриття вихідного файлу для виведення результатів розрахунку за допомогою *оператору* `throw` та *блоку* `try catch`.
- 11) По завершенню розрахунків всіх значень функції запропонувати користувачу з застосуванням оператору *циклу* `while` вибір: завершити роботу програми чи запустити її на повторний розрахунок. Вихід із циклу організувати за допомогою *оператору переходу* `break`.

Теоретичні відомості

Тема 8.1. Функції

8.1.1 Физначення та виклик функції

Функція (function) – це іменований блок коду. Запуск цього коду на виконання здійснюється при виклику функції. Функція може отримувати будь-яку кількість аргументів і (зазвичай) повертає результат. Функція може бути перевантажена, відповідно, те ж ім'я може відноситися до кількох різних функцій.

Визначення функції (function definition) зазвичай складається з *типу повернутого значення* (return type), імені, переліку *параметрів* (parameter) і *тіла функції*. Параметри визначаються в розділеному комами переліку, укладеному в круглі дужки. Дії, які виконує функція, визначаються у блоці операторів, що зветься *тілом функції* (function body).

Для запуску коду функції використовується *оператор виклику* (call operater), що являє собою пару круглих дужок. Оператор виклику отримує вираз, який є функцією чи покажчиком на функцію. В круглих дужках через кому розміщується перелік *аргументів* (argument). Аргументи використовуються для ініціалізації параметрів функції. Тип викликаного виразу – це тип значення, яке повертає функція.

Для прикладу створимо функцію обчислення факторіалу заданого числа. Факторіал числа n є добутком чисел від 1 до n . Дану функцію можна визначити наступним чином:

```
int fact(int val)
{
    int ret = 1; //локальна змінна для збереження результатів в ході
                //їх обчислення
    while (val>1)
```



```
{
    ret *= val--; //ret = ret*val, val= val-1
}
return ret; //повернення результату роботи функції
}
```

Функції присвоєне ім'я `fact`. Вона отримує один параметр типу `int` і повертає значення типу `int`. В циклі `while` обчислюється факторіал з використанням складеного оператора присвоєння та постфіксного оператору декрименту. Оператор `return` виконується в кінці функції і повертає значення змінної `ret`.

Для виклику функції `fact()`, потрібно надати їй значення типу `int`. Результатом виклику також буде значення типу `int`.

```
int main()
{
    int j = fact(5); //j = 120, тобто результату fact(5)
    cout << "Факторіал п'яти 5! = " << j << endl;
    return 0;
}
```

Виклик функції виконує дві дії: ініціалізує параметри функції відповідними аргументами і передає керування коду функції. При цьому виконання функції, *яка викликає* (calling) призупиняється і починається виконання *викликаної* (called) функції.

Виконання функції завершується оператором `return`. Як і виклик функції, оператор `return` виконує дві дії: повертає значення (якщо воно є) і передає керування назад функції, *яка викликає*.

8.1.2 Аргументи та перелік параметрів функції

Аргументи – це ініціалізатори для параметрів функції. Перший аргумент ініціалізує перший параметр, другий аргумент ініціалізує другий параметр і т.д. Тип кожного аргументу має співпадати з типом відповідного параметру, як і тип ініціалізатора має співпадати з типом об'єкту, який він ініціалізує. Потрібно передати точно таку ж кількість аргументів, скільки у функції параметрів.

Перелік параметрів функції може бути пустим, проте не може бути відсутнім. При визначенні функції без параметрів зазвичай використовують пустий список параметрів. Список параметрів, як правило, складається з розділеного комами переліку параметрів, кожен з яких виглядає як одиночне об'явлення. Навіть коли типи двох параметрів однакові, об'явлення слід повторити:


```
int f1(int v1, int v2) { /* ... */ }
int f2(int v1, v2) { /* ... */ } //Помилка!!!
```

Параметри не повинні мати однакові імена.

В якості типу повернутого значення функції застосовується більшість типів. Зокрема, типом повернутого значення може бути `void`, це означає, що функція *не повертає значення*. Типом повернутого значення *не може бути* масив чи функція. Проте функція може повертати *показчик* на масив чи функцію.

8.1.3 Область видимості імен

В мові C++ ім'я має область видимості, а об'єкт – тривалість існування (object lifetime).

- *Область видимості імен* – це частина тексту програми, в якій ім'я відомо.
- *Тривалість існування об'єкту* – це час при виконанні програми, коли об'єкт існує.

Тіло функції, як блок операторів, формує *нову область видимості*, в якій можна визначати змінні. Параметри та змінні, які визначені в тілі функції, називають *локальними змінними* (local variable). Вони є локальними для даної функції і *приховують* (hide) об'явлення того ж імені у зовнішній області видимості. Локальні змінні мають пріоритет над глобальними, об'явленими поза межами блоку з тим же ім'ям.

Об'єкти, які визначені поза межами будь-якої з функцій, існують протягом виконання програми. Такі об'єкти створюються при запуску програми і не видаляються до її завершення. Тривалість існування локальної змінної залежить від того, як вона визначена.

Об'єкти, які відповідають звичайним локальним змінним, створюються у місці визначення змінної у функції (блоці) і видаляються, коли процес виконання досягає кінця блоку. Об'єкти, що існують тільки під час виконання блоку, в якому вони були визначені, називають *автоматичними об'єктами* (automatic object).

По завершення виконання блоку значення автоматичних об'єктів, які були створені в цьому блоці, невизначені.

Параметри – це автоматичні об'єкти. Значення неініціалізованих локальних змінних вбудованого типу невизначені.

Для створення локальної змінної, тривалість існування якої не переривається між викликами функції, при визначенні локальної змінної використовують ключове слово `static`. Кожен *локальний статичний об'єкт* (local static object) ініціалізується раніше, ніж хід виконання досягне

визначення об'єкту. Локальна статична змінна не видаляється по завершенню роботи функції; вона видаляється по завершенню роботи програми.

```
static size_t i = 0; //значення змінної зберігається між викликами функції
```

Як і будь-яке інше ім'я, ім'я функції має бути об'явлене раніше, ніж його можна буде використовувати. Подібно до змінних, функція може бути *визначена* тільки один раз, проте *об'явлена* може бути багаторазово.

8.1.4 Оголошення функції

Об'явлення функції подібно до її визначення, але у об'явлення немає *тіла функції*. В об'явленні тіло функції замінюється крапкою з комою. Оскільки в об'явленні немає тіла функції, *відпадає необхідність* в іменах параметрів. Проте імена параметрів у об'явленні функції часто використовують для полегшення розуміння користувачем призначення функції.

```
void print(vector<int>::const_iterator beg, vector<int>::const_iterator end);
```

Три елементи об'явлення функції: тип повернутого значення, ім'я функції та тип параметрів – описують *інтерфейс* (interface) функції. Вони задають всю інформація, що необхідна для виклику функції. Об'явлення функції називають також *прототипом функції* (function prototype).

Об'явлення функцій розміщують у файлах *заголовку*, а *визначення* – у файлах *вихідного коду*. Файл вихідного коду, в якому функція *визначена*, повинен підключати *заголовок*, в якому функція *об'явлена*. Так компілятор зможе перевірити відповідність визначення та об'явлення.

У файлі *вихідного коду*, де передбачається *використання зовнішньої функції*, також необхідно *підключити заголовок*, в якому ця функція об'явлена. Через файл заголовку з об'явленням функції здійснюється зв'язок між файлом вихідного коду, в якому функція визначена, та файлом вихідного коду, де функція використовується. Відповідно, заголовок з об'явленням функції включається в обидва файли вихідного коду: файл з визначенням функції та файл, в якому вона застосовується. Створені файли заголовку та файли вихідного коду потрібно *підключати* до проекту щоб компілятор їх включив у програму.

Мова C++ дозволяє розділяти програми на логічні частини, надаючи засіб, відомий як *роздільна компіляція* (separate compilation). Роздільна компіляція дозволяє поділити програму на кілька файлів, кожен з яких може бути відкомпільований окремо.

8.1.5 Передача параметрів за посиланням та за значенням

При кожному виклику функції її параметри створюються заново. Параметри ініціалізуються так само, як і звичайні змінні. Якщо параметр – посилання, то параметр прив'язується до свого аргументу. В іншому випадку, значення аргументу копіюється.

Коли параметр – посилання, говорять що аргумент *передається за посиланням* (pass by reference) або що функція *викликається за посиланням* (call by reference). Подібно будь-якому іншому посиланню, параметр посилання – це лише *псевдонім* об'єкту, до якого він прив'язаний, тобто параметр посилання – це псевдонім свого аргументу.

```
void reset(int i) //i - копія аргументу
{
    i = 0; //змінює значення в середині функції, значення переданого
          //аргументу за межами функції не міняється
}
```

Коли значення аргументу копіюється, параметр і аргумент – незалежні об'єкти. Говорят, що такі аргументи *передаються за значенням* (pass by value) або що функція *викликається за значенням* (call by value).

```
void reset(int &i) //i - лише інше ім'я об'єкту
{
    i = 0; //змінює значення об'єкту, на який посилається i
          //у функції змінюється значення зовнішнього об'єкту
}
```

Функція може повертати лише *одне значення*. У випадках, коли функція має повертати більше ніж одне значення, використовують додаткові *параметри посилання*. Так як посилання – лише інше ім'я об'єкту, то змінюючи такі додаткові параметри всередині функції, ми міняємо значення об'єктів, на які вони посилаються, і за її межами. Тобто повертаємо змінені значення переданих за посиланням аргументів в тіло програми по завершенню виконання функції.

Якщо параметр, який передається у функцію не повинен мінятися, потрібно визначати його як *константне посилання* за допомогою ключового слова `const`.

```
void print(const double &y)    //Визначення функції для друку значення
                               //аргументу
{
    static int i = 0;          //Змінна, яка зберігає своє значення між
                               //викликами функції
    cout << "Ітерація " << ++i << endl;
```



```
    cout << "y = " << y << endl;  
}
```

Використання неконстантних посилань обмежує можливість використання функції. Через константний параметр посилання до функції можна передати як константний, так і неконстантний аргумент. Через неконстантний параметр посилання можна передати лише неконстантний аргумент і не можна передавати константні аргументи, в тому числі рядкові чи чисельні літерали.

Функції, які розташовані в одній області видимості, називають *перевантаженими* (overload), якщо вони мають однакові імена, але різні списки параметрів. При виклику такої функції компілятор приймає рішення про використання певної версії на основі типу переданого аргументу. У перевантажених функцій має однозначно відрізнятися тип параметрів, або має бути різна кількість параметрів.

```
void print(const double &y);  
void print(const int *beg, const int *end);
```



Приклад програмної реалізації

Приклад роботи з функціями користувача

```
//Програма обчислення значень функції F(x,y) у заданих точках

#include "stdafx.h"
#include <iostream>
#include "windows.h"
#include "HeaderFxy.h"           //Підключення власного файлу заголовку,
                                //де оголошена структура даних
#include "fanctFxy.h"           //Підключення власного файлу заголовку,
                                //де оголошена функція f

#include <vector>
using std::vector;
using std::cin;
using std::cout;
using std::endl;

int main()
{
    //Необхідно для введення в консолі української мови
    SetConsoleCP(1251);
    //Необхідно для виведення в консолі української мови
    SetConsoleOutputCP(1251);
    while (true)
    {
        //Оголошення та ініціалізація константної змінної
        const double sgX = 1.9;
        const double sgY = 0.2;
        //Оголошення змінної типу створеної структури даних fxy
        fxy fi;
        //Оголошення вектору для збереження змінної типу fxy
        vector<fxy> vectF;
        size_t n;
        cout << "Задайте початкове значення x(0)" << endl;
        cin >> fi.x;
        cout << "Задайте початкове значення y(0)" << endl;
        cin >> fi.y;
        cout << "Задайте кількість розрахунків значень функції F(x,y)"
             << endl;
        cin >> n;
        for (size_t i = 0; i < n; i++)
        {
            //Виклик зовнішньої функції розрахунку залежності
            functXY(fi);
            vectF.push_back(fi); //Запис змінної типу fxy у вектор
            printFXY(vectF[i]); //Виклик зовнішньої функції для
                                //друку елементу вектору
        }
    }
}
```



```
        fi.x += sgX;  
        fi.y += sgY;  
    }  
    cout << "Запустити розрахунок ще раз? [y] [n]" << endl  
        << endl;  
    char y;  
    cin >> y;  
    if (y == 'n')  
    {  
        //Переривання роботи програми шляхом виходу  
        //з циклу while  
        break;  
    }  
}  
return 0;  
}
```

C++

Приклад визначення функцій користувача

```
//FunctXY.cpp
```

```
#include "stdafx.h"
#include <math.h>
#include <iostream>
#include "HeaderFxy.h"           //Підключення власного файлу заголовку,
                                //де оголошена структура даних
#include "fanctFxy.h"           //Підключення власного файлу заголовку,
                                //де оголошені функції

using std::cin;
using std::cout;
using std::endl;

void functXY(fxy &f)           //Визначення функції для розрахунку значення
                                //на окремій ітерації
{
    f.f = (log(sqrt(f.x)) - log(sqrt(f.y)))*pow((f.x - f.y), 1 / 3)
        / (1 / tan(f.x));
}

void printFXY(const fxy &f)     //Визначення функції для друку
                                //елементів вектору
{
    static int i = 0;           //Змінна, яка зберігає своє значення
                                //між викликами функції
    cout << "Ітерація " << ++i << endl;
    cout << "F(" << f.x << "," << f.y << ")=" << f.f << endl;
}
```


Приклад оголошення функцій користувача у файлі заголовку

```
//functFxy.h
```

```
#pragma once
#ifndef FANCTFXY_H //Попередження повторного визначення класу
#define FANCTFXY_H

void functXY(fxy &f); //Оголошення функції functXY із параметром
//посиланням, яка не повертає значення

void printFXY(const fxy &f); //Оголошення функції printFXY із
//параметром константне посиланням,
//яка не повертає значення

#endif FANCTFXY_H
```



Приклад оголошення структури даних користувача у файлі заголовку

```
//HeaderFxy.h
```

```
#pragma once
```

```
#ifndef HEADERFXY_H
```

```
#define HEADERFXY_H
```

```
struct fxy
```

```
{
```

```
    double x;
```

```
    double y;
```

```
    double f;
```

```
};
```

```
#endif HEADERFXY_H
```

//Попередження повторного визначення класу

//Створення власної структури даних (класу)

C++

Завдання до комп'ютерного практикуму №8

Розробити програму (відповідно своєму варіанту завдання), яка реалізує поставлену задачу у зовнішній функції (функціях) користувача. Зберігати результати розрахунку та вхідні дані за допомогою власної структури даних. Розрахунки на кожній ітерації зберігати у вектор. Створити зовнішню власну функцію для виведення елементів вектору на екран та у зовнішній текстовий файл. Передбачити в програмі обробку виключних ситуацій.

При організації функції передбачити друк доданків (співмножників) та поточної суми (добутку) на кожному кроці в залежності від побажання користувача.

Якщо знаменник доданка (співмножника) дорівнює нулю, такий доданок (співмножник) не враховувати та надрукувати відповідне повідомлення зі значенням аргументу.

Налаштувати програму та перевірити її роботу. Продемонструвати роботу програми викладачу.

Варіант	Добуток, який треба розрахувати	Варіант	Сума, яку треба розрахувати
1	$P = \prod_{j=m}^K \frac{j^2 - 3j + 1}{(j-4)(j-2)} \cdot (a-3)$	2	$S = \sum_{n=J}^K \frac{(n+3)^3}{(n-5)^2} \cdot (a+1)$
3	$P = \prod_{n=z}^K \frac{n+2}{(n-\alpha)(n-\beta)} \cdot (a+2)$	4	$S = \sum_{k=L}^M \frac{3k^2}{4(k+1)(k-2)} \cdot (a-4)$
5	$P = \prod_{i=L}^m \frac{2i+1}{(i-3)(i+1)} \cdot (a+3)$	6	$S = \sum_{n=K}^M \frac{n+11}{(n-6)(n-5)} \cdot (a-1)$
7	$P = \prod_{l=i}^j \frac{l^2}{(l-3)(l-\varphi)} \cdot (a-5)$	8	$S = \sum_{i=K1}^N \frac{i+2}{(i-3)(i-5)} \cdot (a+4)$
9	$P = \prod_{k=m}^n \frac{k+8}{(k^2-9k+21) \cdot (a-4)}$	10	$S = \sum_{n=K}^I \frac{(n+\beta)^2}{(n-4)(n-8)} \cdot (a-3)$
11	$P = \prod_{l=M}^N \frac{l^2+3l}{(l-4)(l+5)} \cdot (a+4)$	12	$S = \sum_{i=K}^M \frac{(i+1)^3}{(i^2-i-7) \cdot (a-5)}$
13	$P = \prod_{i=M}^N \frac{i^2+2i+3}{i-3} \cdot (a-2)$	14	$S = \sum_{n=i}^K \frac{n}{n^2-5n+6} \cdot (a-4)$
15	$P = \prod_{k=i}^j \frac{k^2+4k+7}{(k-5)} \cdot (a+1)$	16	$S = \sum_{j=Z1}^{Z2} \frac{(j+2)^3}{(j-8)(j-k)} \cdot (a+3)$
17	$P = \prod_{k=M}^N \frac{k+3}{(k-5)(k-6)} \cdot (a-3)$	18	$S = \sum_{n=L}^K \frac{n-2}{n^2-16} \cdot (a+5)$
19	$P = \prod_{k=M}^L \frac{k+1}{(k-8)(k-6)} \cdot (a-1)$	20	$S = \sum_{i=K}^M \frac{(i+1)^2}{(2i^3-3i-11) \cdot (a-4)}$
21	$P = \prod_{n=z}^K \frac{n}{(n-2)(n-5)} \cdot (a+2)$	22	$S = \sum_{n=K}^L \frac{(n+4)^3}{(n^3-n^2+2n-3) \cdot (a-3)}$

23	$P = \prod_{k=N}^M \frac{k+1}{(k-5)(k-7)} \cdot (a-2)$	24	$S = \sum_{n=L}^k \frac{n^2 - n}{(n^2 + n + 6) \cdot (a+4)}$
25	$P = \prod_{n=M}^K \frac{n}{(n-3)(n-8)} \cdot (a+5)$	26	$S = \sum_{i=M}^k \frac{i^2 - 4i + 7}{(i^2 - 6i + 13) \cdot (a+2)}$
27	$P = \prod_{n=L}^m \frac{(n+3)^3}{(n-5)^2} \cdot (a-3)$	28	$S = \sum_{i=K}^N \frac{i^2 - i + 3}{i-5} \cdot (a+1)$
29	$P = \prod_{l=i}^j \frac{(l+3)^2}{(l-7)(l-9)} \cdot (a-4)$	30	$S = \sum_{i=K}^J \frac{i(i+1)}{(i+5)(i-3)} \cdot (a-2)$

C++

Контрольні питання

- 1) Що таке функція?
- 2) Що таке визначення функції, тип повернутого значення, параметри, тіло функції?
- 3) Що таке оператор виклику функції, з чого він складається?
- 4) Для чого потрібен оператор return?
- 5) Що таке аргументи функції? Для чого використовуються аргументи?
- 6) Що таке перелік параметрів функції? Як об'являються параметри функції? Чи може бути функція без параметрів? Чи може функція не повертати значення? Наведіть приклади.
- 7) Що таке область видимості імен та тривалість існування об'єкту?
- 8) Що таке локальні змінні? Чим вони відрізняються від глобальних?
- 9) Що таке автоматичний об'єкт? Що таке локальний статичний об'єкт? Наведіть приклади?
- 10) Що таке оголошення функції? Що таке інтерфейс функції? Наведіть приклад.
- 11) Як створити зовнішню функцію? Як підключити зовнішню функцію користувача для використання у файлі вихідного коду?
- 12) Що таке роздільна компіляція та для чого вона потрібна?
- 13) Чим відрізняється передача аргументу за посиланням від передачі аргументу за значенням?
- 14) Скільки значень може повертати функція? Чи може функція повернути кілька значень, якщо так, то як це реалізувати? Наведіть приклади.
- 15) Як потрібно визначати параметр функції, який не повинен змінюватись? Наведіть приклади.
- 16) Що таке перевантажені функції? Які особливості їх використання? Наведіть приклади.

Комп'ютерний практикум №9

Класи

Мета: вивчити принципи створення власних *класів* та роботи з ними: *визначення класу*; створення *конструктору класу*; створення *інтерфейсу класу*; створення *реалізації класу*; *оголошення функцій-членів класу*; *виклик функції-членів класу*; створення змінної типу власного класу; передача аргументів *функціям-членам* класу; створення програм з обчислювальною частиною, яка реалізована у власному класі та його *функціях-членах*.

Завдання: створити консольний додаток для розрахунку інтегралу методом Сімпсона згідно отриманого варіанту завдання.

Загальні вимоги.

- 1) Створити консольний додаток з ім'ям виду *PrizvischeKP9*.
- 2) Програмний код модуля має бути чітко структурований.
- 3) Імена об'єктів мають нести сенсові навантаження.
- 4) Програмний код має супроводжуватись коментарями в тексті програми.

Вимоги до виконання.

- 1) Створити за допомогою *файлу заголовку* власний *клас даних*, який реалізуватиме знаходження заданого інтегралу методом Сімпсона чи за формулою трапецій (згідно варіанту).
- 2) Введення вхідних даних реалізувати в основному тілі програми.
- 3) Всі етапи обчислення реалізувати у власному класі.
- 4) У класі передбачити три конструктори: конструктор за замовчуванням; конструктор для ініціалізації класу всіма необхідними даними (коефіцієнти рівняння, межі інтегрування, точність інтегрування); для ініціалізації класу лише межами інтегрування (в реалізації класу передбачити ініціалізацію коефіцієнтів рівняння, меж інтегрування та точності інтегрування значеннями за замовчуванням);
- 5) У класі створити інтерфейс класу (відкрити для користувача частину) та реалізацію класу (інкапсульовану всередині класу його частину). Інтерфейсна частина класу та частина його реалізації повинні мати не менше однієї функції-члену кожна.
- 6) У програмі передбачити можливість вибору користувача: вводити всі вхідні дані (коефіцієнти рівняння, межі інтегрування, точність інтегрування) для ініціалізації об'єкту класу за допомогою розширеного конструктора;

вводити лише інтервал інтегрування для виклику конструктора класу, який передає ініціалізацію інших вхідних даних значеннями за замовчуванням у блоці реалізації класу.

- 7) Передбачити виведення результатів розрахунку на кожній ітерації, а також знайденого інтегралу з заданою точністю на екран та у зовнішній файл.

Теоретичні відомості

Тема 9.1. Класи

9.1.1 Абстракція даних та інкапсуляція

Класи в мові C++ використовуються для визначення власних типів даних. Фундаментальними поняттями концепції класів є абстракція даних та інкапсуляція.

Абстракція даних (data abstraction) – програмний підхід, що заснований на розділенні інтерфейсу та реалізації. *Інтерфейс* (interface) класу складається з операцій, які користувач класу може виконати з його об'єктом. *Реалізація* (implementation) включає змінні-члени класу, тіла функцій, що складають інтерфейс, а також інші функції, котрі потрібні для визначення класу, проте не призначені для загального використання.

Інкапсуляція (encapsulation) забезпечує розділення інтерфейсу та реалізації класу. Інкапсульований клас приховує свою реалізацію від користувачів, які можуть використовувати інтерфейс, проте не мають доступу до реалізації класу.

Під *користувачами* класу в C++ маються на увазі розробники програмного коду, які використовують вже створений раніше іншими розробниками клас.

Клас, який використовує абстракцію даних та інкапсуляцію, називають *абстрактним типом даних* (abstract data type). Програмісти, які працюють із класом, не мають знати як внутрішньо працює цей тип. Вони можуть розглядати його як абстракцію.

Для забезпечення інкапсуляції в C++ використовують *специфікатори доступу* (access specifier).

- Члени класу, які визначені після *специфікатора public*, доступні для всіх частин програми. *Відкриті члени* (public member) визначають *інтерфейс класу*.
- Члени, які визначені після *специфікатора private*, є *закритими членами* (private member), вони доступні для функцій-членів класу, але не доступні для коду, який цей клас використовує. Розділи *private* інкапсулюють (приховують) реалізацію.

Визначення класу виглядає наступним чином:


```
class MyClass //Ім'я класу
{
public:
    //Інтерфейс класу
private:
    //Реалізація класу
};
```

При визначенні класу можна використовувати ключові слова `struct` або `class`. Відмінність у визначенні класу за допомогою цих ключових слів полягає у заданому за замовчуванням *рівні доступу*. Якщо використовується *ключове слово* `struct`, то члени, які визначені до першого специфікатора доступу, будуть *відкритими* (`public`), тобто входять до інтерфейсу класу. Якщо використовується *ключове слово* `class`, то члени, які визначені до першого специфікатора доступу, будуть *закритими* (`private`), тобто входять до реалізації класу.

Інкапсуляція надає дві важливі переваги.

- Код користувача не може за необачності пошкодити інкапсульований об'єкт.
- Реалізація інкапсульованого класу може з часом змінитися і це не вимагатиме змін у коді на рівні користувача.

9.1.2 Змінні-члени та функції-члени

Змінні, які входять у визначення класу, називають *змінними-членами* (`data member`).

Члени класу, які є функціями, називають *функціями-членами* (`member function`).

Функції-члени визначають та об'являють як звичайні функції. Функції-члени *мають бути* оголошені в класі, проте визначені вони *можуть бути* безпосередньо в класі або поза тілом класу. Функції, які не є членами класу, але є складовими інтерфейсу, об'являються та визначаються поза класом, але в тому ж самому файлі що і клас.

```
class MyClass
{
public: //Інтерфейс класу
    void Method(); //Оголошення функції-члену класу без параметрів
private: //Реалізація класу
    float k1 = 1; //Ініціалізація даних-членів класу
    float k2 = 1;
    float k3 = 1;
    float a=-10;
    float b=10;
```



```

double MyFunc(const double&); //Оголошення функції-члену класу
                                //з параметром - константним посиланням
MyClass& MyFunc2(const MyClass&) //Оголошення функції-члену класу
                                //з параметром - константним посиланням на тип класу
}; //Завершення тіла класу

void MyClass::Method() //Визначення функції-члену класу без параметрів
{
    double za, zb;
    za = MyFunc(a);
    zb = MyFunc(b);
}

double MyClass::MyFunc(const double &x) //Визначення функції-члену класу
{
    //з параметром - константним посиланням поза тілом класу
    double z;
    z = k1*exp(k2*x) + k3*x;
    return z;
}

```

Ім'я функції-члена, яка оголошена в тілі класу, але визначена поза тілом класу, повинне включати ім'я класу, якому вона належить:

```
void MyClass::Method() //Визначення функції-члену поза тілом класу
```

Ім'я функції `MyClass::Method()` використовує оператор області видимості, щоб вказати, що дана функція оголошена в межах класу `MyClass`.

9.1.3 Оператор доступу до члену та оператор області видимості

Кожен клас визначає власну область видимості. Поза *областю видимості класу* (`class scope`) до звичайних даних і функцій його члени можуть звертатись лише через об'єкт, посилання або покажчик, використовуючи *оператор доступу до члену* (`.`). Для доступу до членів типу з класу використовується *оператор області видимості* (`::`). В будь-якому випадку наступне за оператором ім'я має бути членом відповідного класу.

У функції-члені можна безпосередньо звернутися до членів об'єкту, з якого вона була викликана. Для цього використовується покажчик `this`. *Параметр `this`* визначається неявно та автоматично і його можна використовувати в тілі функції-члену.

```

MyClass& MyClass::MyFunc2(const MyClass &rhs) //Визначення функції-члену
{
    //класу з параметром - константним посиланням поза тілом класу
    a += rhs.a;
    return *this; //повертає об'єкт, для якого була викликана функція
}

```


Автори класів інколи визначають *допоміжні функції*, які використовуються як частина інтерфейсу класу, але при цьому членами класу вони не є. Такі функції об'являються (але не визначаються) в тому ж заголовку, що і сам клас після визначення класу. Таким чином, щоб використовувати будь-яку частину інтерфейсу класу, користувачу достатньо підключити лише один файл заголовку.

9.1.4 Конструктори

Кожен клас визначає, як можуть бути ініціалізовані об'єкти його типу. Клас контролює ініціалізацію об'єкту за рахунок визначення однієї чи декількох спеціальних функцій-членів, відомих як *конструктори* (constructor). Задача конструктора – ініціалізувати змінні-члени об'єкту класу. Конструктор виконується кожен раз, коли створюється об'єкт класу.

Ім'я конструктора *співпадає* з іменем класу. На відміну від інших функцій, у конструкторів *немає типу* повернутого значення. Як і інші функції, конструктори мають список параметрів (можливо пустий) і тіло (можливо пусте). У класа може бути *декілька* конструкторів. Подібно будь-якій іншій перевантаженій функції, конструктори мають *відрізнятися* один від одного *кількістю* або *типами* своїх параметрів. Конструктори, на відміну від інших функцій, не можуть бути оголошені константами.

Якщо в класі не визначено жодного конструктору, класи самі контролюють ініціалізацію відкритих змінних-членів значеннями за замовчуванням, визначаючи спеціальний конструктор, що зветься *стандартним конструктором* (default constructor). Стандартним вважається конструктор, який не отримує ніяких аргументів. У такому випадку змінні-члени ініціалізуються внутрішньокласовими ініціалізаторами чи значеннями за замовчуванням.

Якщо клас не визначає конструктори явно, компілятор сам визначить стандартний конструктор неявно. Створений компілятором конструктор зветься *синтезованим стандартним конструктором* (synthesized default constructor).

Класи, члени яких мають *вбудований* чи *складений тип* (відповідно не мають ініціалізатора за замовчуванням), можуть розраховувати на синтезований стандартний конструктор, *тільки якщо у всіх* таких членів є внутрішньокласові ініціалізатори.

```
struct MyStruct
{
    int *pi = 0;    //Внутрішньокласовий ініціалізатор змінної
                  //складеного типу
    float y = 1;    //Внутрішньокласовий ініціалізатор змінної
                  //вбудованого типу
```



```
double a;           //Визначення змінної-члену вбудованого типу
                    //без ініціалізатора
}; //Завершення тіла класу
```

Якщо у класа визначений хоча б один конструктор, то компілятор не буде створювати автоматично стандартний конструктор. У такому разі стандартний конструктор потрібно обов'язково визначити в класі самостійно.

Якщо в класі є інші конструктори, то можна попросити компілятор створити *стандартний конструктор* автоматично. Для цього після списку параметрів вказується частина = default.

```
MyClass() = default;
```

Оскільки цей конструктор не має параметрів – він є стандартним конструктором. Застосування цього конструктора можливе лише у випадку, коли для змінних-членів вбудованих та складених типів є *внутрішньокласові ініціалізатори*.

При створенні конструктору класу після переліку параметрів ставиться двокрапка, за якою розміщується *перелік ініціалізації конструктора* (constructor initializer list), який визначає вихідні значення для однієї чи декількох змінних-членів створюваного об'єкту. Завершується конструктор фігурними дужками, які містять тіло конструктора. *Ініціалізатор конструктору* – це перелік імен змінних-членів класу, кожне з яких супроводжується вихідним значенням у круглих (або фігурних) дужках. Якщо ініціалізацій кілька, вони відділяються комами.

```
MyClass() = default; //Стандартний конструктор
MyClass(const float &a1, const float &b1) : a(a1), b(b1) {};
//Конструктор класу
MyClass(const float &kof1, const float &kof2, const int &n, const float
&a1, const float &b1) : k1(kof1), k2(kof2), k3(kof2*n), a(a1), b(b1) {};
//Конструктор класу
```

У даному прикладі першим іде стандартний конструктор, який для ініціалізації змінних членів при створенні об'єкту класу використовує внутрішньокласові ініціалізатори (якщо вони є), в іншому разі використовує ініціалізацію змінних-членів значеннями за замовчуванням (окрім вбудованих та складених типів).

Другим іде конструктор класу, який ініціалізує змінні-члени a та b параметрами a1 та b1 відповідно. Змінні члени k1, k2 та k3 будуть ініціалізовані внутрішньокласовими ініціалізаторами.

Третій конструктор ініціалізує всі змінні-члени створюваного об'єкту класу переліком ініціалізації конструктора, який розміщений між двокрапкою та

фігурними дужками. При цьому змінна-член k3 ініціалізується добутком параметрів kof2 та n.

Зазвичай для конструктора краще використовувати внутрішньокласовий ініціалізатор, якщо він є і присвоює члену класу вірні значення. Якщо компілятор не підтримує внутрішньокласову ініціалізацію (введена в стандарті C++11), кожен конструктор повинен явно ініціалізувати кожен член вбудованого типу.

У наведених вище конструкторів тіла пусті. Єдине їх призначення – присвоїти значення змінним-членам. Якщо нічого іншого робити не потрібно, то тіло функції залишається пустим.

Як і інші функції члени, конструктори можуть визначатися як у тілі класу, так і за його межами. В тілі класу визначають функції, які складаються не більше як з одного-двох операторів, щоб не ускладнювати розуміння класу. Функції-члени, в тому числі конструктори, які мають більший обсяг коду у своєму тілі, об'являються в тілі класу, а визначаються одразу після нього в тому ж самому файлі заголовку. Перед іменем конструктора, який визначений за межами класу, як і перед іменем інших функцій-членів, вказується ім'я класу та оператор області видимості.

```
MyClass::MyClass()  
{  
    //Тіло конструктору  
}
```

Ім'я класу перед іменем функції вказується для того, щоб зазначити що дана функція відноситься до вказаного класу. Оскільки ім'я функції співпадає з іменем класу, значить ця функція є конструктором зазначеного класу.

9.1.5 Дружні відносини

Клас може дозволити іншому класу чи функції отримати доступ до своїх закритих членів, встановивши для них *дружні відносини* (friend). Клас об'являє функцію дружньою, включивши її оголошення з ключовим словом friend.

```
class MyClass  
{  
    friend float add(const float&);  
    friend std::ostream &print(std::ostream&, const MyClass&);  
public: //Інтерфейс класу  
    MyClass() = default; //Стандартний конструктор  
    MyClass(const float &a1, const float &b1) : a(a1), b(b1) {};  
    //Конструктор класу  
    MyClass(const float &kof1, const float &kof2, const int &n, const  
float &a1, const float &b1) : k1(kof1), k2(kof2), k3(kof2*n), a(a1),  
b(b1) {};
```



```
void Method(); //Оголошення функції-члену класу без параметрів
private: //Реалізація класу
float k1 = 1; //Ініціалізація даних-членів класу
float k2 = 1;
float k3 = 1;
float a=-10;
float b=10;
double MyFunc(const double&); //Оголошення функції-члену класу
                                //з параметром - константним посиланням
MyClass& MyFunc2(const MyClass&) //Оголошення функції-члену класу
                                //з параметром - константним посиланням на тип класу
}; //Завершення тіла класу
//Оголошення частин, що не є складовими інтерфейсу класу
float add(const float&);
std::ostream &print(std::ostream&, const MyClass&);
```

Оголошення друзів може розташовуватись лише у визначенні класу, використовуватись у класі вони можуть будь-де. Друзі не є членами класу і не підкоряються специфікатору доступу розділу, в якому вони оголошені. Оголошення друзів бажано групувати на початку чи в кінці визначення класу.

Оголошення дружніх відносин встановлює лише права доступу. Це не оголошення функції. Для можливості виклику дружньої функції користувачами класу її слід також оголосити. Для доступу користувачів до дружніх функцій їх зазвичай оголошують поза класом, у тому ж заголовку, що і сам клас.

Приклад програмної реалізації

Приклад роботи з класами (Частина 1)

```
// Програма пошуку кореня рівняння методом половинного ділення
//

#include "stdafx.h"
#include "MyMethod.h"
#include "windows.h" //Необхідно для виведення в консолі української мови
using std::cout;
using std::cin;
using std::endl;

int main()
{
    SetConsoleCP(1251); //Необхідно для виведення в консолі
                        //української мови
    SetConsoleOutputCP(1251); //Необхідно для виведення в консолі
                              //української мови

    float k1, k2, k3, a, b, e;
    cout << "Введіть коефіцієнти рівняння" << endl;
    cout << "Введіть коефіцієнт k1" << endl;
    cin >> k1;
    cout << "Введіть коефіцієнт k2" << endl;
    cin >> k2;
    cout << "Введіть коефіцієнт k3" << endl;
    cin >> k3;
    cout << "Задайте ліву межу інтервалу пошуку кореня рівняння a"
        << endl;
    cin >> a;
    cout << "Задайте праву межу інтервалу пошуку кореня рівняння b"
        << endl;
    cin >> b;
    cout << "Задайте точність пошуку кореня рівняння e" << endl;
    cin >> e;
    MyClass MyCalc(k1, k2, k3, a, b, e);
    MyCalc.Method();
    system("pause"); // Команда затримки екрану
    return 0;
}
```


Приклад створення класу користувача у файлі заголовку

// MyMethod.h створений клас для пошуку кореня рівняння методом половинного ділення

```
#include "math.h"
#include <iostream>
using std::cin;
using std::cout;
using std::endl;

class MyClass
{
public: //Інтерфейс класу
    //Конструктор класу за замовчуванням
    MyClass() = default;
    //Конструктор класу
    MyClass(const float &k1, const float &k2, const float &k3,
            const float &a1, const float &b1, const float &e1) :
        A(a1), B(b1), E(e1), K1(k1), K2(k2), K3(k3) {};
    //Конструктор класу
    MyClass(const float &a1, const float &b1) : A(a1), B(b1) {};
    void Method(); //Оголошення функції-члену класу без параметрів

private: //Реалізація класу
    float K1 = 1; //Ініціалізація даних-членів класу
    float K2 = 1;
    float K3 = 1;
    float A=-100;
    float B=100;
    float E=0.001;
    double MyFunc(const double&); //Оголошення функції-члену класу
    //з параметром - константним посиланням
};

void MyClass::Method() //Визначення функції-члену класу без параметрів
{
    double za, zb, zx, x;
    double a(A), b(B); //Ініціалізація змінних a та b значеннями
    // змінних A та B відповідно

    unsigned i=0;
    za = MyFunc(A);
    zb = MyFunc(B);
    cout << "Знаходження коренів рівняння методом половинного ділення"
    << endl;
    if (za*zb<0)
    {
        do
        {
```



```
        ++i;
        x = (a + b) / 2;
        zx = MyFunc(x);
        cout << "Ітерація " << i << ", x=" << x << ", z="
              << zx << endl;
        za * zx < 0 ? b = x : a = x, za = zx;
    } while (abs(b-a)>E);
    cout << "Корінь рівняння x = " << x << endl;
}
else
{
    cout << "Корінь на інтервалі відсутній. Перевірте
           відокремлення коренів!" << endl;
}
}

double MyClass::MyFunc(const double &x) //Визначення функції-члену класу
                                         //з параметром – константним посиланням
{
    double z;
    z = K1*exp(K2*x) + K3*x;
    return z;
}
```


Приклад роботи з класами (Частина 2)

```
// QuadraticEquation.cpp: определяет точку входа для консольного
приложения.
//

#include "stdafx.h"
#include <iostream>
#include <fstream>
#include <string>
#include "QuadraticEquation.h"
#include "windows.h" //Необхідно для виведення в консолі української мови

using namespace std;

int main()
{
    //Необхідно для виведення в консолі української мови
    SetConsoleCP(1251);
    //Необхідно для виведення в консолі української мови
    SetConsoleOutputCP(1251);
    float a, b, c;
    char ans = 'y', ind_p;

    do
    {
        cout << "Ведіть коефіцієнти квадратного рівняння:" << endl;
        cout << "a = ";
        cin >> a;
        cout << "b = ";
        cin >> b;
        cout << "c = ";
        cin >> c;
        //Ініціалізуємо примірник класу QuadraticEquation
        QuadraticEquation MyEquation(a, b, c);
        cout << "Необхідний запис результатів у файл? (y/n) ";
        cin >> ind_p;
        if (ind_p == 'y')//Виводити результати до файлу чи на екран
        {
            string rezalt_file;
            cout << "Вкажіть і'мя файлу - ";
            cin >> rezalt_file;
            ofstream outfile(rezalt_file, ofstream::app);
            //Проводимо розрахунок та виводимо результати на екран
            MyEquation.Method();
            //Виводимо результати до файлу
            MyEquation.SolutionPrint_toFile(outfile);
        }
    }
```



```
else
{
    //Проводимо розрахунок та виводимо результати на екран
    MyEquation.Method();
}
cout << endl << "Наступне рівняння? (y/n)";
cin >> ans;
} while (ans == 'y');//Повторити розрахунок для іншого рівняння?
return 0;
}
```

C++

Приклад створення класу користувача у файлі заголовку

```
#pragma once
// QuadraticEquation.h клас, створений для розв'язання квадратних рівнянь
//
#include "stdafx.h"
#include <iostream>
#include <fstream>
#include <math.h>

using namespace std;

class QuadraticEquation //Оголошення класу
{
public://Інтерфейс класу (відкриті члени)
    QuadraticEquation() = default;
    QuadraticEquation(const float& a, const float& b, const float& c) :
        A(a), B(b), C(c) {};
    void SolutionPrint_toFile(ofstream &outf);
    void Method();

private://Реалізація класу (закриті члени)
    int ind;
    float X, Y, Z;
    double x1, x2, d;
    float A = 1;
    float B = 1;
    float C = 1;
    void SolutionRezalt();
};//Кінець оголошення класу

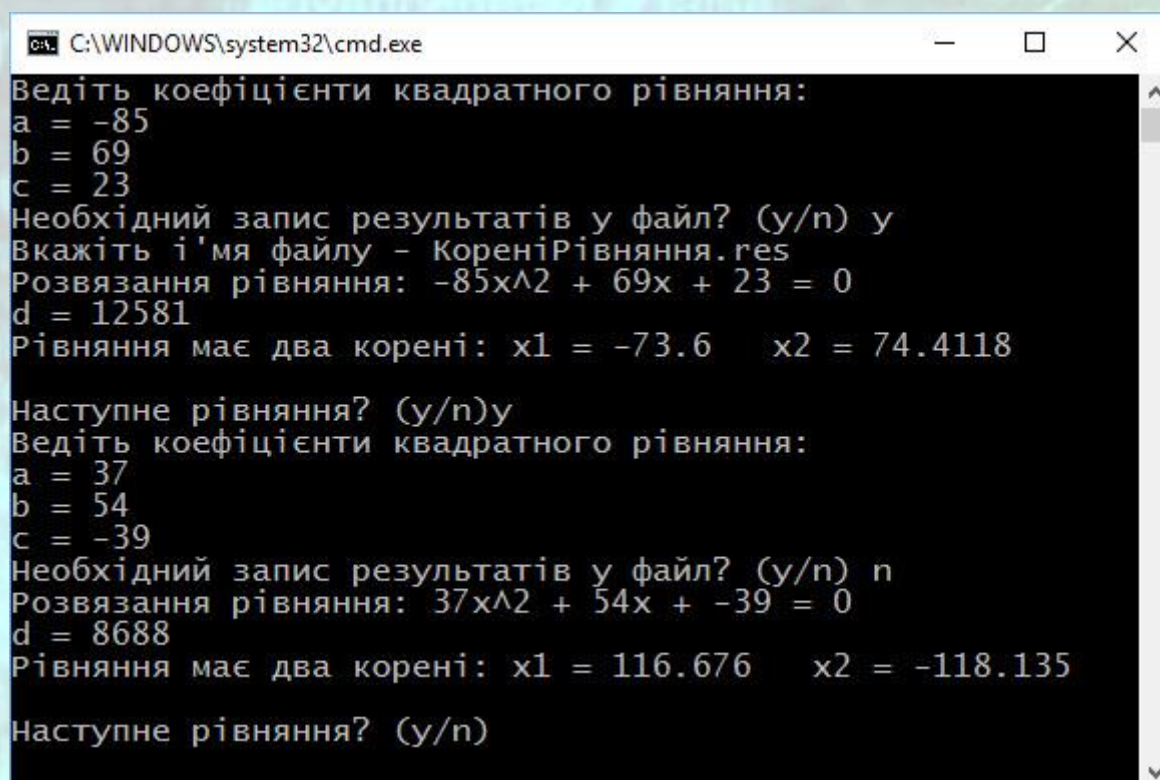
//Визначення функції-члену класу, яка знаходить кореня рівняння
void QuadraticEquation::Method()
{
    d = B * B - 4 * A * C;
    if (d < 0)
    {
        ind = 0;
    }
    else
    {
        if (d == 0)
        {
            x1 = -B / A / 2;
            ind = 1;
        }
        else
        {
            x1 = (-B + d) / (2 * A);
            x2 = (-B - d) / (2 * A);
        }
    }
}
```



```
        ind = 2;
    };
    SolutionRezalt();
    return;
};

//Визначення функції-члену класу, яка виводить результати на екран
void QuadraticEquation::SolutionRezalt()
{
    cout << "Розв'язання рівняння: " << A << "x^2 + " << B << "x + "
        << C << " = 0" << endl;
    cout << "d = " << d << endl;
    switch (ind)
    {
    case 0:
        cout << "Немає дійсних коренів" << endl;
        break;
    case 1:
        cout << "Рівняння має один корінь: x = " << x1 << endl;
        break;
    case 2:
        cout << "Рівняння має два корені: x1 = " << x1 << "    x2 = "
            << x2 << endl;
        break;
    }
}

//Визначення функції-члену класу, яка друкує результати у файл
void QuadraticEquation::SolutionPrint_toFile(ofstream &OSTR)
{
    OSTR << "Розв'язання рівняння: " << A << "x^2 + " << B << "x + "
        << C << " = 0" << endl;
    OSTR << "d = " << d << endl;
    switch (ind)
    {
    case 0:
        OSTR << "Немає дійсних коренів" << endl;
        break;
    case 1:
        OSTR << "Рівняння має один корінь: x = " << x1 << endl;
        break;
    case 2:
        OSTR << "Рівняння має два корені: x1 = " << x1 << "    x2 = "
            << x2 << endl;
        break;
    }
}
```


Результати розрахунку

```
C:\WINDOWS\system32\cmd.exe
Ведіть коефіцієнти квадратного рівняння:
a = -85
b = 69
c = 23
Необхідний запис результатів у файл? (y/n) y
Вкажіть і'мя файлу - КореніРівняння.res
Розв'язання рівняння:  $-85x^2 + 69x + 23 = 0$ 
d = 12581
Рівняння має два корені:  $x_1 = -73.6$   $x_2 = 74.4118$ 

Наступне рівняння? (y/n)u
Ведіть коефіцієнти квадратного рівняння:
a = 37
b = 54
c = -39
Необхідний запис результатів у файл? (y/n) n
Розв'язання рівняння:  $37x^2 + 54x + -39 = 0$ 
d = 8688
Рівняння має два корені:  $x_1 = 116.676$   $x_2 = -118.135$ 

Наступне рівняння? (y/n)
```


Файл з результатами розрахунку КореніРівняння.res

Розв'язання рівняння: $-85x^2 + 69x + 23 = 0$

$d = 12581$

Рівняння має два корені: $x_1 = -73.6$ $x_2 = 74.4118$



Завдання до комп'ютерного практикуму №9

Обчислити інтеграл для парних варіантів за формулою трапецій, а для непарних варіантів за формулою Сімпсона з точністю до 0,001. Величину кроку h , що забезпечує задану точність, визначити за допомогою подвійного перерахунку.

1. $\int_a^b \frac{dx}{\sqrt{2x^2+3,4}}$

2. $\int_a^b e^{-3x^2+x+0,5} dx$

3. $\int_a^b \frac{dx}{\sqrt{3x^3+2}}$

4. $\int_a^b \frac{\ln(3+x^2)}{3+2,5x^2} dx$

5. $\int_a^b \sqrt{\frac{1-0,35x^2}{1-0,5x^2}} dx$

6. $\int_a^b e^{-5x^2+3x+1} dx$

7. $\int_a^b \sqrt{\frac{2-0,3x}{3-2x^2}} dx$

8. $\int_a^b \sqrt{1,3+4,2x^3} dx$

9. $\int_a^b \frac{\ln(2+x)}{1,2x} dx$

10. $\int_a^b \frac{\ln(1,5+\sqrt{x})}{\sqrt[3]{\sqrt{x+1}}} dx$

11. $\int_a^b \frac{\ln(0,3x+2,5)}{\sqrt{x^2+0,8}} dx$

12. $\int_a^b \sqrt{x+1} \lg(x+3,4) dx$

13. $\int_a^b \frac{xdx}{\sqrt{1,4x^2+0,7}}$

14. $\int_a^b \frac{\cos x}{x+2} dx$

15. $\int_a^b \frac{dx}{\sqrt{x^2-0,9}}$

16. $\int_a^b \sqrt{2x} \cos x^2 dx$

17. $\int_a^b \frac{\ln(x^2+0,9)}{x-1} dx$

18. $\int_a^b \frac{dx}{\sqrt{2x^2+0,5x}}$

19. $\int_a^b \sqrt{\frac{2x+0,5}{1+0,2x^2}} dx$

20. $\int_a^b (1,8x+0,7) \sin x dx$

21. $\int_a^b \ln(1+x^2) dx$

22. $\int_a^b e^{-0,5x^2} \sin(0,7x) dx$

23. $\int_a^b \frac{\sqrt{1,1+x^2}}{1+\cos(1,1x)} dx$

24. $\int_a^b \frac{e^{-x^2} \sin(0,5x)}{0,6+x^2} dx$

25. $\int_a^b \frac{\ln(x^2+2)}{2x+1} dx$

26. $\int_a^b \frac{dx}{\sqrt{(1-x^2)(1-0,35x^2)}}$

$$27. \int_a^b \frac{\ln(3,5 + \sqrt[3]{x^2})}{\sqrt[3]{x+3x}} dx$$

$$29. \int_a^b \frac{e^{3x^2+2}}{3x+1} dx$$

$$28. \int_a^b \frac{\sqrt{1,3+4,2x^3}}{3(x^2-2x)} dx$$

$$30. \int_a^b \frac{\sqrt{x^3+1}}{\lg(x+3,4)} dx$$

A large, stylized logo for C++ programming language. The 'C' is a thick, rounded letter. The '+' is a thick, rounded plus sign. The second '+' is a thick, rounded plus sign. The 'C' and the first '+' are white, while the second '+' is blue. The background is a dark blue gradient with a faint, glowing green and blue pattern that resembles a stylized 'C++' or a network of connections.

Контрольні питання

- 1) Що таке абстракція даних? Що таке абстрактний тип даних?
- 2) Що таке інтерфейс та реалізація класу?
- 3) Що таке інкапсуляція? Що таке специфікатори доступу?
- 4) У чому різниця між використанням ключових слів `struct` та `class`?
- 5) Що таке функції-члени? Де та як вони об'являються та визначаються?
- 6) Що таке область видимості класу? Наведіть приклад визначення функції-члену поза тілом класу.
- 7) Для чого використовується параметр `this`? Як звернутися до члену об'єкту класу? Наведіть приклади.
- 8) Як підключити допоміжні функції для використання класом без їх включення у клас?
- 9) Що таке конструктор, для чого він використовується?
- 10) Що таке стандартний конструктор? Що таке синтезований стандартний конструктор? Як об'явити стандартний конструктор?
- 11) Що таке перелік ініціалізації конструктору? Для чого він потрібен?
- 12) Де потрібно визначати конструктор? Із чого складається конструктор? Наведіть приклади оголошення та визначення конструкторів?
- 13) Що таке дружні відносини? Як об'явити дружню функцію? Як зробити дружню функцію доступною для користувачів класу?

Рекомендована література

Базова

- 1) Стенли Б. Липпман, Жози Лажойе, Барбара Э. Му Язык программирования C++. Базовый курс. М.: Вильямс, 2014. – 1120 с.
- 2) Страуструп, Бьярне. Программирование: принципы и практика использования C++, 2-е изд. - М.: ООО «И.Д. Вильямс», 2016. – 1328 с.
- 3) Прата С. Язык программирования C++. Лекции и упражнения. Учебник. - СПб.: ООО «ДиаСофтЮП», 2005. 1104 с.
- 4) Мейерс С. Эффективное использование C++. 50 рекомендаций по улучшению ваших программ и проектов. - М.: Питер-ДМК, 2006. – 240 с.

Допоміжна

- 5) Аверкин В.П., Бобровский А.И. и др. под ред. Хомоненко А.Д. Программирование на C++. Учебное пособие. Корона-Принт, 1999. – 252 с.
- 6) Александреску, Андрей. Современное проектирование на C++. Серия C++ In-Depth, т.3.: Пер. с англ. – М.: Вильямс, 2002. – 336 с.
- 7) Астахова, И.Ф., Власов, С.В. Язык C++. Учебное пособие. И.Ф. Астахова, С.В. Власов, В.В. Фертиков, А.В. Ларин. – Мн.: Новое знание, 2003. – 203 с.
- 8) Седжвик, Роберт. Фундаментальные алгоритмы на C++. Анализ/Структуры данных/Сортировка/Поиск: Пер. с англ./ Роберт Седжвик. – К.: ДиаСофт, 2001. – 688 с.
- 9) Седжвик, Роберт. Фундаментальные алгоритмы на C++. Алгоритмы на графах: Пер. с англ./ Роберт Седжвик. – СПб.: ДиаСофтЮП, 2002. – 496 с.
- 10) Дейтел Х.М., Дейтел П.Дж. Как программировать на C++: Пятое издание. Пер. с англ. – М.: ООО «Бином-Пресс», 2008. - 1456 с.

Інформаційні ресурси

- 12) Язык программирования C++ [Электрон. ресурс] // Основы программирования на языках Си и C++ для начинающих - Режим доступа: <http://cppstudio.com/cat/274/>

- 13) Бендюг Владислав Іванович [Електрон. ресурс] // Офіційний сайт – Режим доступу: <http://бендюг.укр/>
- 14) Єдине інформаційне середовище - Національний технічний університет України «КПІ» [Електрон. ресурс] // Електронний КАМПУС НТУУ «КПІ» – Режим доступу: <http://login.kpi.ua/>



Додаток А

Формули для довідок

1. Рівняння Менделєєва-Клапейрона:

$$P \cdot V = \frac{m}{M} \cdot R \cdot T,$$

де P – тиск; V – об'єм; m – маса газу; M – молекулярна маса; R – універсальна газова стала; T – абсолютна температура.

2. Співвідношення між елементами трикутника зі сторонами a , b , c та протилежними кутами A , B , C .

2.1. Теорема синусів:

$$\frac{a}{\sin A} = \frac{b}{\sin B} = \frac{c}{\sin C}$$

де R – радіус описаного кола.

2.2. Теорема косинусів:

$$a^2 = b^2 + c^2 - 2bc \cos A.$$

2.3. Площа трикутника:

$$S = \frac{a h_a}{2}$$

де h_a висота трикутника.

2.4. Форма Герона:

$$S = \sqrt{p(p-a)(p-b)(p-c)}, \quad \left(p = \frac{a+b+c}{2}\right).$$

2.5.

$$S = \frac{1}{2} a b \sin c.$$

2.6.

$$S = r \cdot p,$$

r – радіус кола, вписаного у трикутник.

2.7. Для рівнобічного трикутника:

$$S = \frac{a^2 \sqrt{3}}{4}.$$

3. Площа фігур.

3.1. Паралелограма:

$$S = b h,$$

де b – сторона, h – висота.

3.2. Ромба:

$$S = \frac{1}{2} d_1 d_2$$

де d_1 та d_2 – діагоналі.

3.3. Трапеції:

$$S = \frac{a+b}{2} h,$$

де a та b – сторони, h – висота.

$$S = m h,$$

де m – середня лінія.

3.4. Квадрата:

$$S = a^2,$$

де a – сторона.

3.5. Прямокутника:

$$S = a b,$$

де a та b – сторони.

3.6. Кола:

$$S = \pi R^2,$$

де R – радіус.

4. Об'єми різних фігур:

де S – площа основи, H – висота.

4.1. Призми:

$$V = S H.$$

4.2. Циліндра:

$$V = S H.$$

4.3. Піраміди:

$$V = \frac{1}{3} \dot{S} \cdot H$$

4.4. Конуса:

$$V = \frac{1}{3} \cdot \dot{S} \cdot H,$$

де

$$\dot{S} = \pi R^2$$

4.5 Кулі:

$$V = \frac{4}{3} \pi R^3$$

де R – радіус.

5. Квадратні рівняння:

$$5.1. \quad \begin{aligned} a x^2 + b x + c &= 0; \\ x_{1,2} &= \frac{-b \pm \sqrt{b^2 - 4 a c}}{2 a}. \end{aligned}$$

$$5.2. \quad \begin{aligned} x^2 + p x + q &= 0; \\ x_{1,2} &= -\frac{p}{2} \pm \sqrt{\frac{p^2}{4} - q}. \end{aligned}$$

5.3. Теорема Вієта:

коли $x^2 + p \cdot x + q = 0$, то $x_1 + x_2 = -p$, $x_1 \cdot x_2 = q$.

6. Рішення системи рівнянь з двома невідомими:

$$\text{Для } \begin{cases} a_1 \cdot x + b_1 \cdot y = c_1, \\ a_2 \cdot x + b_2 \cdot y = c_2, \end{cases} \quad \begin{aligned} \Delta &= a_1 \cdot b_2 - a_2 \cdot b_1; \\ \Delta_x &= c_1 \cdot b_2 - c_2 \cdot b_1; \\ \Delta_y &= a_1 \cdot c_2 - a_2 \cdot c_1. \end{aligned}$$

$$\text{Тоді } x = \frac{\Delta_x}{\Delta}; \quad y = \frac{\Delta_y}{\Delta}$$

7. Довжина відрізка АВ ($A(x_1, y_1), B(x_2, y_2)$),

$$l_{AB} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}.$$

8. Якщо точка $D(x, y)$ знаходиться у площі трикутника ABC, де $A(x_1, y_1), B(x_2, y_2), C(x_3, y_3)$, то

$$S_{ABC} = S_{ADC} + S_{CDB} + S_{ADB}.$$